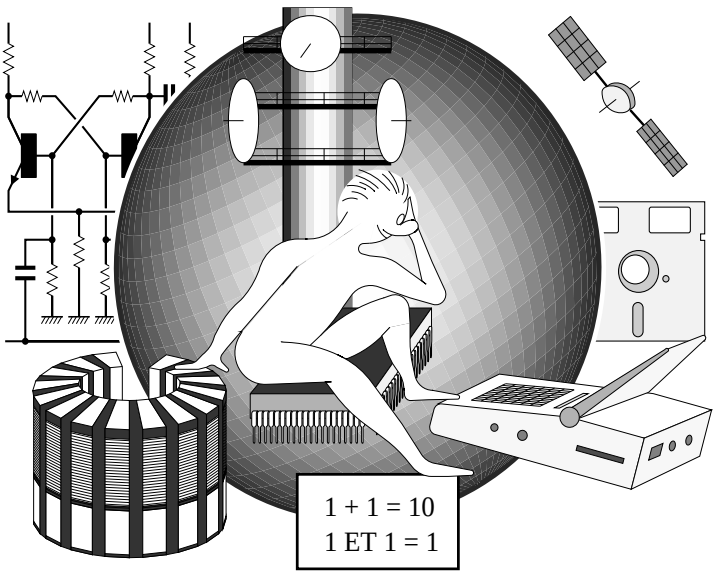
Pierre BLANC

INITIATION A L'INFORMATIQUE

Préface du Professeur Néel,
Membre de l'Institut,
Prix Nobel de Physique.



Dessins humoristiques de Benoît Blanc

Publié en libre service

Cet ouvrage, rédigé dans un langage simple, un style clair et précis, poursuit un double objectif :

- donner à un large public, et en particulier aux jeunes, un minimum de connaissances sur l'informatique, une vue synthétique des tenants et des aboutissants de cette discipline qui a bouleversé notre espace culturel et économique,

- fournir les bases indispensables à tout candidat à la programmation des ordinateurs ou à l'emploi rationnel des outils informatiques, spécialement ceux relevant de la bureautique ou ceux disponibles sur les ordinateurs personnels.

DROITS DE COPIE (COPYRIGHT)

Dépôt légal: 1er trimestre 1996.

L'auteur est ingénieur au Commissariat à l'Energie Atomique à Grenoble,
Département de Recherches Fondamentales sur la Matière Condensée. Courrier électronique : pblanc@cea.fr

VERSION 1995

TABLE DES MATIERES

PREFACE

En quelques décennies, avec l'aide des physiciens, l'informatique, sur des bases aussi simples qu'une utilisation judicieuse des 0 et des 1, a envahi la plupart des activités des sociétés humaines développées. Elle a multiplié, dans des proportions inimaginées jusque là, la vitesse des procédés conçus pour assister le cerveau, comme dans les calculs arithmétiques, la gestion des stocks, l'accumulation et la transmission à distance des informations ...

Aucune autre révolution industrielle n'a été aussi rapide, ni aussi profonde. Inévitablement, un développement aussi foudroyant s'est effectué dans une certaine anarchie : constructeurs, matériels divers, systèmes d'exploitation, langages et logiciels ont proliféré et offrent à un observateur un peu cartésien un tableau déroutant.

Sans doute, le possesseur d'un micro-ordinateur obtient-il des résultats satisfaisants, et cela devrait lui suffire, en suivant les injonctions de son manuel et en appuyant sur telle ou telle touche de son clavier pour réaliser telle ou telle opération. Il n'a pas besoin de savoir ce qui se passe dans les profondeurs de son unité centrale, de même que la plupart des automobilistes ignorent sans inconvénient les principes des moteurs à explosion et les subtilités de l'avance à l'allumage. Cependant l'honnête homme aimerait en connaître davantage et le spécialiste lui-même, avant de se lancer dans cette activité et d'en devenir expert, a dû éprouver le besoin de savoir en quoi elle consistait.

Si une initiation à l'informatique est hautement souhaitable, elle est malheureusement difficile, car il ne s'agit pas de déduire les conséquences logiques de quelques idées générales, mais d'exposer le fonctionnement d'un nombre élevé de dispositifs ingénieux, étroitement imbriqués les uns dans les autres, selon des principes plutôt empiriques. C'est pourquoi je présente avec plaisir l'*Initiation à l'Informatique* de M. Pierre Blanc qui me semble très réussie. L'auteur n'est pas un spécialiste de l'informatique, mais c'est un physicien expérimentateur de valeur, qui travaille depuis plus de vingt ans, sur un large éventail de sujets, au polygone scientifique de Grenoble, haut lieu de recherche scientifique et technique. L'informatique est pour lui un outil de travail indispensable et il en a suivi tous les développements, depuis les balbutiements initiaux.

Dans cet ouvrage, après deux chapitres consacrés à un rapide historique et à un rappel des bases mathématiques, les trois suivants traitent des machines, de leur architecture et des périphériques. Ils devraient intéresser en priorité les usagers des micro-ordinateurs, ainsi que les deux chapitres qui leur font suite et parlent des objets de l'informatique, mots, nombres, chaînes, fichiers, ..., et du langage machine. Cette première partie, complétée par une étude de la programmation en langages évolués, devrait ainsi lui permettre d'évaluer, par rapport aux autres, les avantages et les inconvénients de son propre langage.

Les cinq chapitres d'une deuxième partie présentent un caractère très général et intéressent n'importe quel lecteur, soucieux de culture générale, en exposant les grandes applications de l'informatique, comme la bureautique où les traitements de textes finiront par supprimer les machines à écrire, la gestion, la conception assistée par ordinateur qui révolutionne le dessin industriel et architectural, les réseaux de communications et bien d'autres. L'auteur se penche finalement sur le rôle de l'informatique dans le monde du travail, dans le secteur tertiaire aussi bien que dans les arts, sans oublier les problèmes juridiques soulevés par une activité originale sans précédent.

Sous une forme accessible, les jeunes, auxquels l'auteur a spécialement pensé, trouveront dans ce livre tous les éléments nécessaires à orienter éventuellement leur avenir dans cette discipline. Un copieux index permet aussi à un lecteur pressé de trouver et de retrouver les points qui l'intéressent et d'interpréter une terminologie ésotérique hésitant entre l'anglais et le français.

Bien entendu, cette Introduction ne peut qu'effleurer un domaine aussi étendu que complexe. Elle donnera certainement l'envie d'en savoir davantage et ce sera peut-être un de ses grands mérites. A cet égard, un index bibliographique permet de s'orienter parmi quelques-uns des nombreux ouvrages parus à ce sujet au cours des dernières années.

Je félicite M. Blanc d'avoir mené à bien une tâche difficile et je souhaite à son livre tout le succès qu'il mérite.

Louis Néel

AVANT-PROPOS

Ce manuel est écrit spécialement à l'intention des lycéens désirant acquérir les premières notions de l'informatique. Le niveau requis pour sa compréhension est à peu près celui de la classe de terminale. L'auteur espère pouvoir accompagner les premiers pas du lecteur sur les chemins de cette discipline d'une manière précise et pas trop rébarbative. Tout adolescent peut légitimement éprouver le désir d'être introduit dans ce monde prodigieux, dont l'impact sur sa vie professionnelle s'annonce, selon les augures, comme des plus prometteurs ou des plus menaçants.

Pourra également tirer profit de cet exposé l'honnête homme, soucieux de vivre avec son temps et désireux de ne pas rester à l'écart de ces nouvelles techniques, dont l'essor fait qualifier les décennies que nous vivons de seconde révolution industrielle. En plus d'une connaissance académique sur le sujet, il devrait acquérir le vocabulaire propre à cette discipline, qui hélas, comme tout vocabulaire technique, handicape le profane dans ses tentatives d'approche. L'auteur lui souhaite de trouver dans ce livre le désir d'utiliser sans plus tarder les remarquables moyens mis à sa disposition par l'informatique.

De même, beaucoup d'utilisateurs d'ordinateurs (quelquefois contraints et forcés) gagneront à cette lecture une meilleure compréhension de leur outil de travail et pourront ainsi par la suite vivre avec lui une relation moins conflictuelle et, – pourquoi pas ? – enrichissante.

Les sujets traités obéissent à un plan qui, partant de généralités, en arrive presque au mode d'emploi des outils les plus populaires. Si les premiers chapitres relèvent de l'informatique générale, les derniers sont nettement orientés vers le monde des calculateurs personnels et vers la bureautique. Cependant, l'étudiant qui aura ainsi assimilé les bases de ce qu'il est convenu d'appeler la micro-informatique ne devra pas sous-estimer son acquit : il aura fait un grand pas vers la compréhension des systèmes les plus complexes. Il pourra ensuite lire des ouvrages spécialisés ou aborder les formations adaptées à ses ambitions.

Ce livre voulant être avant tout culturel, on y expliquera plutôt le pourquoi que le comment. On espère ainsi pouvoir démystifier l'informatique et, tout d'abord, les termes utilisés par ses professionnels. On a pris soin d'employer au maximum un français académique en indiquant au passage les termes anglais ou franglais les plus courants **(1)**. Ces termes seront critiqués, lorsqu'ils sont illogiques, car le manque de rigueur sous-jacent ne contribue pas peu à dresser entre l'informatique et le néophyte un obstacle de premier plan. Cependant, cette discipline ayant créé quelques concepts nouveaux, il est normal d'admettre l'introduction de néologismes, même tirés de langues étrangères, à condition que leur construction obéisse à un minimum de rationalisme.

L'ouvrage est divisé en deux parties inégales. La première, après quelques notions de base, est surtout consacrée au matériel. Le chapitre III sera peut-être un peu difficile à lire par qui n'a pas reçu quelques notions d'électricité, mais il ne sera indispensable qu'à celui qui veut comprendre le fonctionnement des appareils associés à l'informatique.

La seconde partie, la plus importante, est vouée au logiciel. Certains passages pourront être laissés de côté en première lecture, comme la représentation des nombres réels dans le chapitre VI ainsi que la totalité du chapitre VII sur le langage machine. Les notes en bas de page s'adressent quant à elles aux spécialistes de telle ou telle discipline ou bien à ceux qui veulent toujours en savoir plus. A part ces quelques passages, l'ouvrage est surtout descriptif et ne devrait pas présenter de difficultés, y compris le deuxième chapitre (bases du calcul binaire) qui exigera peut-être un effort dont on devra surtout ne pas faire l'économie.

REMERCIEMENTS

L'auteur exprime sa gratitude au Professeur Néel qui a bien voulu lire ce livre, proposer des amendements et lui donner une préface. Il remercie également ceux de ses collègues qui ont accepté de lire une partie de son texte et apporter critiques et suggestions constructives. Il reconnaît l'apport de MM. Jean-Luc Archimbaud (sections relatives à Internet), Constant Axelrad (partie consacrée à l'industrie dans le dernier chapitre) et Luc Billard (annexe 4). Ainsi amendé par ces experts, le livre gagnera en exactitude et crédibilité. Ses remerciements s'adressent également à son épouse Lisette, à son neveu Benoît qui ont bien voulu contribuer aux illustrations, ainsi qu'à Alexandre et à ses parents, grâce auxquels un visage d'enfant éclairera cet ouvrage dédié à la jeunesse.

---

(1) On a imprimé en italique les citations, termes étrangers, mots techniques, ainsi que ceux requérant une attention spéciale de la part du lecteur. Les lettres isolées en italique se réfèrent quant à elles, selon l'usage établi, à une variable mathématique ; les passages en caractères **helvetica**, à des mots ou des expressions faisant partie des ordres qu'on peut donner aux ordinateurs.

Chapitre I

HISTORIQUE

1 - PREHISTOIRE

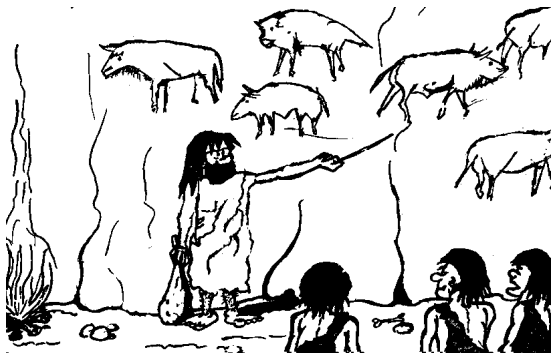
On peut définir l'informatique comme "**la science de l'information**", ce qui englobe sa collecte (sa *saisie* pour les professionnels), son traitement, sa conservation et sa communication. A ce dernier titre, elle est aussi vieille que l'humanité et remonte à l'apparition du langage humain. L'homme primitif a sans doute commencé par émettre des grognements à l'adresse de ses semblables pour leur communiquer ses impressions ou ses sentiments. Le **message** ainsi émis (présence d'un danger, proximité de nourriture...) était déjà précieux, mais il sera beaucoup plus utilisable lorsque de subjectif il est devenu objectif. Pour cela, on a dû certainement comprendre très vite que la meilleure façon de rendre une information objective était de la **numériser**.

On a découvert en Australie une tribu primitive qui ne savait compter que jusqu'à trois : *un, deux, beaucoup*. Ces humains n'ont sans doute pas subi l'épreuve d'une sévère compétition pour la vie, étant incapables du moindre partage de butin et inaptes au dénombrement correct de leurs ennemis. En effet, quand des guetteurs peuvent rapporter : "à l'est, il arrive une centaine d'ennemis" alors que d'autres en annoncent une dizaine à l'ouest, la qualité de telles informations *numériques* **(1)** ne peut que puissamment bénéficier à la stratégie de défense du clan.

Comme l'enfant, l'*homo sapiens* a appris à compter sur ses doigts. Mais ce compte ne va pas loin. On pense que, pour l'étendre, il a d'abord choisi de compter jusqu'à 60. Pourquoi ce nombre ? On l'ignore, mais on peut penser qu'il aurait un rapport avec les cycles de la lune et du soleil (60 jours correspondent environ à 2 lunaisons et un an à 6 fois 60 jours) **(2)**.

Continuer dans cette voie – sans système simplificateur – était sans issue à cause de l'infinité des nombres. Aussi, un sage sans doute eut l'idée de grouper par 60, puis par 60 fois 60,... de petits cailloux représentant les objets à compter. C'était la numération **itérative** à **base 60**. Des vestiges de ce système sont parvenus jusqu'à nous puisqu'on mesure encore de cette façon le temps et les angles (60 secondes font une minute, etc). Cette méthode permet d'évaluer n'importe quel nombre entier et ce, quelle que soit la base. Outre la base 60 et bien sûr la base 10, diverses bases eurent cours ici ou là, comme la base 12, dont il subsiste des traces : on compte les œufs par *douzaines* et naguère par *grosses* (12 douzaines) ; les Anglo-Saxons (bien que ce ne soit plus légal) utilisent encore un système de mesures à base 12.

Après avoir appris à dénombrer, nos ancêtres ont commencé à convertir deux nombres en un seul par une **opération arithmétique**. Les deux opérations fondamentales, addition et soustraction, ont dû être assez vite inventées pour le suivi des troupeaux ou celui des récoltes : mise en commun, pertes, essaimage. Plus tard, multiplication et division ont permis d'abréger une suite d'additions ou de soustractions en une seule opération. On n'a pas trouvé traces de telles connaissances antérieures à la civilisation de **Sumer** (environ 4000 ans AC), c.-à-d. aux plus anciens vestiges de l'écriture.



Un bison, deux bisons, beaucoup de bisons.

Cependant, en admirant les fresques dans les grottes **préhistoriques**, on peut rêver et se demander si de telles compositions, troupeaux ou gravures symboliques, n'auraient pas servi, entre autres, à transmettre aux initiés les connaissances d'alors, dont l'art du dénombrement.

---

(1) Le profane emploierait peut-être *chiffré* pour dire *affecté d'un nombre*. Mais il vaut mieux utiliser pour cela *numérique* et garder à *chiffré* le sens de *codé* (cf. *indéchiffrable*). Les chiffres sont des symboles qui servent à écrire les nombres.

(2) Témoin parmi d'autres, le monument néolithique de Stonehenge (Angleterre) possédait une couronne extérieure formée exactement de 60 mégalithes (30 verticaux et 30 horizontaux).

2 - ANTIQUITE

Les systèmes de numération antiques étaient loin d'être aussi simples que le nôtre, car ils utilisaient plusieurs bases mélangées, le plus souvent **60**, **10** et **5**. A Sumer, c'était plutôt la base **60** ; en **Egypte**, en **Grèce** (à l'époque archaïque), en **Amérique centrale** ou en **Chine**, c'était plutôt **5** et **10**. Ces derniers systèmes ont des traits communs avec celui des **Romains**, beaucoup plus connu : ses symboles, en nombre réduit (I, V, X, L, C, D, M,... pour **1**, **5**, **10**, **50**, **100**, **500**, **1000**...), s'ajoutaient ou se retranchaient entre eux pour former les nombres, selon la règle suivante : les symboles placés par ordre décroissant de gauche à droite s'additionnent, mais tout symbole brisant cet ordre se retranche. On utilise encore ce système de nos jours pour numéroter. Voici quelques exemples et leur équivalent dans notre système :

CM **= 900,** MMCIII **= 2103,** MCMLXXXIX **= 1989.**

Les Grecs de l'époque classique avaient un système bien à eux, peut-être élégant, mais pas plus simple, dans lequel les lettres de l'alphabet représentaient des nombres : α, β,...θ pour les unités, ι, κ, λ, ... pour les dizaines, des apostrophes pour les centaines et les milliers. On peut retenir de ce survol rapide des systèmes de numération antiques qu'ils n'étaient pas faciles et exigeaient de leurs utilisateurs beaucoup d'apprentissage – d'où sûrement, une sélection sévère. Les initiés retiraient de leurs efforts un pouvoir plus ou moins occulte, qu'il s'agisse des chamans primitifs, des mages babyloniens ou des scribes égyptiens. De tout temps d'ailleurs, la connaissance des nombres, de leur propriétés ou de leur manipulation a confiné à la **magie**.

3 - LES PROGRES DECISIFS

C'est l'**invention du zéro** qui a fait franchir le plus grand pas à la numération *itérative*, car elle a permis la **notation de position stricte**. Bien que le zéro ait été plus ou moins connu en Egypte et en Chaldée, on attribue aux **savants hindous** son invention vers l'an 400 de notre ère, ainsi que celle de la notation de position (vers 600). Les navigateurs **arabes** ont importé des Indes, chez eux d'abord, puis au 12e siècle en Europe, ces notions nouvelles et les symboles qui les accompagnaient (les *chiffres arabes)*.

Sans le zéro en effet, impossible d'adopter le système très simple qui est maintenant le nôtre, dans lequel on classe les éléments par groupes de *N*, puis de N^2, N^3 unités, ... et on représente le nombre obtenu par une suite ordonnée de chiffres dont la position indique la classe du groupement. Dans le système décimal, tout nombre *M* s'écrit "... *jklmn*" où *j,k,l,m,n* = 0,1,2,3,...8,9. Cette écriture symbolique signifie que *M* est égal à :

$$M = ...10^4\, j + 10^3\, k + 10^2\, l + 10^1\, m + 10^0\, n$$

Exemple : 1995 =

$$1\times10^3 + 9\times10^2 + 9\times10^1 + 5\times10^0 = 1000 + 900 + 90 + 5$$

Ces propos peuvent prêter à sourire et faire penser à la *prose de Monsieur Jourdain*. Il faut pourtant bien comprendre ce symbolisme pour apprendre à compter dans un système à base non décimale.

Cette notation entraîne une conséquence capitale, la simplicité des opérations. Toutes obéissent maintenant à des **règles très simples** *(algorithmes)*, qui s'appliquent successivement à tous les chiffres exprimant le nombre. Essayez de trouver une règle pour multiplier deux nombres romains, par exemple LXIV par LXXVI, dont le produit est MMMMDCCCLXIV : vous saisirez vite le mérite de notre notation **(3)**!

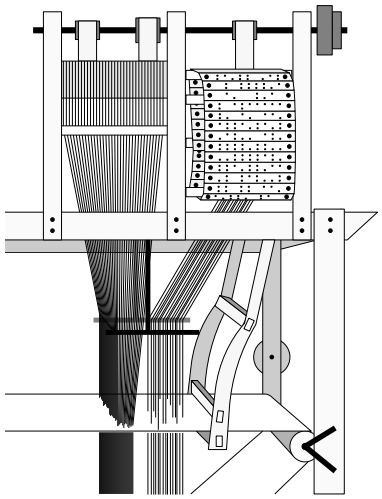
4 - LA MECANISATION

Cette simplification a permis une étape ultérieure, la **mécanisation des calculs**. Le boulier chinois est sans doute le plus lointain ancêtre des machines à calculer et on l'utilise encore, bien qu'il ne calcule pas vraiment, mais constitue plutôt une aide à la mémoire de l'opérateur. Les 16e et 17e siècles voient éclore en Europe des **machines à rouages** de plus en plus complexes, dont celle de Schikard, celle de **Pascal** (addition-soustraction, 1642) et celle de Leibniz (multiplication-division). Elles reposent sur le principe de l'engrenage à échappement : chaque fois qu'une roue dentée a tourné de dix pas, elle déclenche l'avance d'un pas de la roue adjacente. De telles machines ont trôné dans nos bureaux et nos boutiques jusque dans les années 1970.

Parallèlement, peut-être de tous temps, des mécaniciens créaient des **automates**, soit pour les besoins de l'horlogerie (les *jacquemarts* par exemple), la distraction des princes, l'étude de la biologie ou dans un but utilitaire. Héritiers des Grecs (Héron

(3) Ces opérations exigeaient des abaques. Le lecteur qui voudrait en savoir plus visitera avec intérêt le Palais de la Découverte à Paris, département des Mathématiques.

d'Alexandrie entre autres), des Byzantins, des Italiens (Léonard de Vinci)..., ils ont fabriqué des mécanismes étonnants pour leur époque. Le plus célèbre de ces créateurs est **Vaucanson** (vers 1740), mais il ne faut pas oublier Falcon (vers 1730), Bouchon, Jaquet-Droz, Jacquard (vers 1800), ...



Métier à la Jacquard **(4)**

On remarquera en haut à droite les grandes cartes perforées disposées en chapelet. Elles contiennent le programme fixant le dessin du jacquard et commandent le mouvement des "lisses".

Ces pionniers se sont vite aperçus qu'un automate est d'autant plus intéressant qu'il peut varier ses effets. Les mouvements qu'il doit accomplir sont alors **programmés** sur un support amovible : arbre à cames, tambour à picots (comme dans les pianos mécaniques), chapelet de cartes perforées (métier à tisser de **Jacquard**, orgues de Barbarie ...).

On attribue à **Babbage** le mérite d'avoir songé, vers 1850, à doter les machines à calculer des mêmes avantages que les automates en concevant une calculatrice avec un programme *écrit* sur **ruban perforé** et donc modifiable **(5)**. Mais c'est **Hollerith** qui, vers 1880, construisit la première **machine programmable** utilitaire : elle servit à recenser la population des USA. Il fondera une entreprise qui, quelques années plus tard, deviendra **IBM** *(International Business Machines)* **(6)**.

Jusqu'ici, il s'agit toujours de systèmes à rouages avec des rapports de 10 entre roues successives ; le calcul est donc décimal. Mais l'électricité et en particulier les électro-aimants commencent à imposer leur supériorité dans diverses applications (le télégraphe par exemple). Or ces appareils n'ont que deux états stables ; leur fonctionnement est beaucoup mieux décrit par le **calcul binaire** (cf. chap. II) heureusement introduit fort à propos par **Boole** à partir de 1847.

5 - L'ELECTRONIQUE

Les **tubes électroniques**, peu après leur découverte (1918), seront vite utilisés dans les machines à calculer qui progresseront spectaculairement en rapidité (3 millisecondes pour une multiplication). Des chercheurs se pencheront dans les années 1915-1950 sur les aspects théoriques posés par ces appareils. **Torres y Quevedo** montrera la supériorité du calcul numérique binaire, **Couffignal** s'intéressera aux procédés de résolution de problèmes non solubles par l'algèbre, **Turing** fera de ce calcul une analyse qui aboutira à une conception abstraite, mais fertile, la *machine de Turing*. **Von Neumann** concevra l'architecture encore utilisée sur la quasi-totalité des machines actuelles, à savoir **un seul opérateur** *(processeur)* réalisant en séquence toutes les fonctions (entrées, sorties, calcul, rangement) à l'aide d'**une seule mémoire** où sont placés indifféremment programmes et données.

La deuxième guerre mondiale donne un coup de fouet à ces développements. Un énorme calculateur *(computer)* de 18 000 tubes, l'**ENIAC**, est mis en service aux USA pour les besoins de la défense en 1943-45 (30 tonnes, 160 m^2, 150 kW). Il sera suivi par d'autres modèles de moins en moins monstrueux ; leur diffusion restera toutefois modeste (quelques milliers d'exemplaires, mais de types divers).

L'arrivée du **transistor** (1948) inaugurera une étape nouvelle, car ce *composant électronique* va balayer le tube, tant il lui est supérieur :

– il est beaucoup plus fiable (la durée des tubes n'atteint pas 10 000 h),
– il consomme beaucoup moins (1000 fois moins qu'un tube),
– il est beaucoup moins encombrant et plus robuste,
– très vite, il sera plus facile à construire et bien moins cher qu'un tube.

Les calculateurs à transistors (de la taille d'une armoire) vont alors essaimer de par le monde, tant leur emploi devient aisé et leurs prestations prodigieuses. Le marché est nettement dominé par IBM (par exemple la série des 360, 1964), mais d'autres firmes soutiennent la concurrence : citons Remington (Univac), Bull, Burroughs, Control Data, Honeywell, Ferranti, Digital Equipment, ...

(4) Encore en activité, ce métier à tisser entièrement manuel est visible à la *Maison des Canuts* à Lyon, Croix-Rousse.
(5) Babbage avait pour collaboratrice ***Ada***, ou plus exactement Adélaïde, fille du poète Byron, comtesse de Lovelace, dont on reparlera.
(6) Depuis lors, IBM, souvent surnommé *BigBlue* ou bien *le géant d'Armonk*, est numéro un mondial de l'informatique.

6 - LA MICRO-ELECTRONIQUE

L'étape suivante démarre avec l'apparition du *circuit intégré* ou **puce** *(chip* en anglais), dans laquelle les ingénieurs implantent plusieurs centaines, milliers, puis millions de transistors. Toute l'unité centrale (cf. chap. IV) d'un calculateur va tenir dans une puce. On va alors assister au déferlement des **calculettes** dans les bureaux, où elles détrôneront les *règles à calcul* d'antan. Ce marché est dominé par **Hewlett-Packard**, qui introduira les premières calculettes programmables en 1973. Mais d'autres firmes vont lui emboîter le pas (citons Texas Instruments, Sharp, Canon, Casio, ...) et des appareils de plus en plus étoffés (avec cassettes, imprimantes...) vont voir le jour (dont le *Micral* de la firme française R2E, puis les *ordinateurs familiaux).*

Les puces, issues de la *micro-électronique* (implantation de circuits au micron près), ont valu aux techniques dérivées le nom de *micro-informatique*. Au même moment, naît la **disquette**, support d'information tellement pratique et bon marché qu'il met un terme immédiat au règne presque séculaire des bandes et cartes perforées.

La mécanographie disparaît alors, avec son cortège de machines fragiles et bruyantes : perforatrices, trieuses, reproductrices, ... Vers 1980, IBM, roi dans ce domaine, mais absent jusque-là du créneau des petits calculateurs, lance son "**personal computer**" (ou **PC**) qui, muni d'un vrai système d'exploitation (le MS-DOS, cf. chap. VIII), *fait entrer la micro-informatique dans la cour des grands*. Ce calculateur de bureau, au succès foudroyant, marquera le véritable début de la civilisation informatique.

En effet, non seulement ce PC sera amplement copié par de nombreuses firmes s'évertuant à produire des **compatibles IBM-PC**, mais de plus quelques novateurs créeront des modèles rivaux, parfois mieux adaptés à des situations données (comme Atari et surtout **Apple** avec ses **Macintosh**). Le PC s'améliorera au fil des ans (version XT, puis AT, AT386, 486, Pentium). La concurrence sera sévère entre fabricants de *compatibles* et les prix calculés au plus juste. Les **logiciels** (programmes) écrits pour eux vont couvrir un champ d'application de plus en plus vaste, de sorte que ces techniques vont inonder les entreprises durant la décennie 80-90.

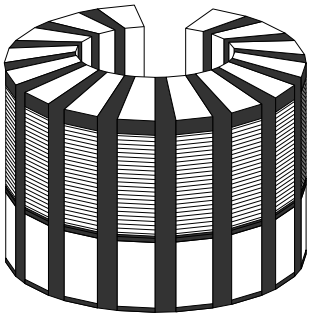
Quant à l'**informatique lourde** (par opposition à l'informatique *personnelle*), si elle n'a pas subi pareille révolution, elle a néanmoins poursuivi son développement en bénéficiant de la miniaturisation des circuits de base. On peut la diviser en trois secteurs principaux :

– celui de l'informatique "**départementale**" au service de quelques dizaines d'employés (c'est la *mini-informatique* du jargon professionnel) ;

– celui de la "**station de travail**", spécialisée, très technique, souvent vouée au dessin (CAO, traitement d'images...), à la disposition d'un petit groupe de concepteurs ;

– celui des **calculateurs géants** dans lequel IBM, longtemps meneur, s'est fait ravir le *ruban bleu* par **Cray** (il n'y a que quelques unités de ce type de machine par grand pays, chacune pouvant servir à des centaines d'utilisateurs)**(7)**.

Calculateur Cray II.



La compétition reste vive entre ces grands secteurs. Cependant, sans vouloir jouer au prophète, on peut parier sur la poussée inéluctable du secteur *personnel* et, sinon le déclin des autres, du moins leur nécessité de composer avec les petits nouveaux venus, beaucoup mieux adaptés que les grosses machines au foisonnement du génie humain. En particulier, l'informatique départementale, synonyme autrefois de calcul centralisé, fait place à la notion de *calcul réparti* sur des PC autour d'un noyau central, lequel se borne bien souvent à n'être qu'un *serveur*, serveur de fichiers et de programmes accessibles à tous via un réseau de communication. Quant aux calculateurs géants, ils restent indispensables à la résolution de très gros problèmes, à cause de leur puissance inégalée, puissance due à des concepts nouveaux, différents de celui de Von Neumann, comme celui de *calcul parallèle*.

On verra dans l'avant-dernier chapitre non seulement comment l'**industrie** s'est laissé *subjuguer* par l'outil informatique, mais aussi comment tous les pans de l'activité humaine se sont vus plus ou moins – et plutôt plus que moins – envahis, perturbés, rajeunis... par l'informatique personnelle. Cette révolution aura été l'une des plus rapides de l'histoire humaine, mais elle ne se déroulera pas sans crises ni sans heurts, péripéties dont nous tenterons de faire une synthèse dans le dernier chapitre.

(7) L'adjectif géant se rapporte plus à la puissance, à la complexité et au prix qu'à l'aspect : utilisant elles-aussi la micro-électronique, ces machines sont devenues très raisonnables en taille (1 m^3 environ pour le Cray-II en volume, 1,1 m en hauteur). Leur forme compacte minimise les délais de propagation des signaux, mais accroît l'échauffement. On les refroidit par circulation forcée d'un liquide.

EXERCICES

Les notes qui suivent constituent, plutôt que des exercices, un rappel de quelques notions, à l'intention des lecteurs pour qui elles ne seraient pas familières.

Puissance d'un nombre :

La notation a^m, qui se lit "a à la puissance m", désigne le produit de a m fois par lui-même (m–1 multiplications) :

$$a^m = a \times a \times ... \times a \times a \qquad (m \text{ fois})$$

m est appelé "exposant" de a. Il en découle les règles suivantes, avec a quelconque :

$a^m \times a^n = a^{m+n}$	(à cause de la définition de a^m)
$a^m / a^n = a^{m-n}$	(pour la même raison)
$a^0 = 1$	(en faisant $m=n$ ci-dessus)
$a^{-n} = 1 / a^n$	(*puisque* $a^0 = 1$)

Ces propriétés, faciles à démontrer pour m et n entiers, peuvent être étendues aux exposants x non entiers. Si $y = a^x$, le nombre réel x est le *logarithme* à base a de y ; il bénéficie de toutes les propriétés des m et n ci-dessus.

Les seules bases de logarithmes utilisées en pratique sont les nombres e = 2,71828... en mathématiques, 10 dans les sciences et techniques usuelles, 2 (et parfois 8, 16, ...) dans les études théoriques sur l'informatique, les probabilités...

Ainsi, on utilise couramment les puissances entières de 10 pour désigner les multiples et sous-multiples d'une grandeur :

$10^2 = 100$, $10^3 = 1000$, $10^6 = 1\,000\,000$, $10^{-3} = 0{,}001$

Préfixes multiplicateurs :

Les préfixes ci-après sont couramment utilisés pour désigner les multiples et sous-multiples d'unités dans le système décimal. La liste ci-dessous donne la valeur du multiplicateur qu'ils représentent, ainsi que leur écriture abrégée.

femto	pico	nano	micro	milli	centi	déci
10^{-15}	10^{-12}	10^{-9}	10^{-6}	10^{-3}	10^{-2}	10^{-1}
f	**p**	**n**	**µ**	**m**	**c**	**d**

déca	hecto	kilo	méga	giga	téra
10^1	10^2	10^3	10^6	10^9	10^{12}
da	**h**	**k**	**M**	**G**	**T**

Ainsi, une microseconde vaut un millionième de seconde, un nanomètre un milliardième de mètre ou un millième de micron (micron est le nom abrégé du micromètre ou millionième de mètre), une kilotonne 10^9 grammes, un mégafranc un million de francs, ...

Exemples plus difficiles, mais faisant déjà partie de la vie courante : un décanewton (daN) vaut 10 newtons, soit environ 1,02 kgf, un peu plus d'un kilogramme-force, ancienne unité de force, et un hectopascal (hPa) cent pascals ou un millibar, unité autrefois légale dans les mesures de pression.

Ces préfixes seront beaucoup employés dans cet ouvrage qui décrit comment, avec des techniques avoisinant l'infiniment petit, on manipule des valeurs qui frôlent l'infiniment grand.

Chapitre II

LE CALCUL BINAIRE

1 - LE SYSTEME DECIMAL

A l'école, nous avons tous appris à compter dans le système décimal. Pour ce faire, dans les campagnes par exemple, il y a encore 50 ans, les instituteurs demandaient aux jeunes écoliers de tailler jusqu'à 200 *bûchettes* dans les buissons ; après quoi, ils leur apprenaient à les grouper par 10, puis par 100...

On peut compter avec un système basé sur n'importe quel nombre. On a vu, dans le premier chapitre, le rôle joué par les systèmes duodécimal (à base 12) et sexagésimal (à base 60). Citons-en un autre, parfois employé après une élection, lorsque les scrutateurs comptabilisent chaque voix en inscrivant une barre dans une figure à cinq branches (par exemple un carré barré en diagonale). Sans en être conscients, comme Monsieur Jourdain, ils se servent alors d'un système à base 5.

Compter dans le système décimal veut dire tout d'abord n'employer que 10 chiffres (nos chiffres arabes), puis grouper les éléments à dénombrer par puissances de 10 c'est-à-dire en paquets de 10, puis de 100 si nécessaire, puis de 1000, etc. Ces groupes possèdent d'ailleurs des noms : dizaines, centaines, milliers... On écrit le résultat en notation de position, c.-à-d. sous forme d'une suite de chiffres, tous inférieurs à 10, égaux successivement au nombre de ..., de milliers, de centaines, de dizaines et enfin d'unités isolées. Par exemple, le nombre 1789, écrit en décimal (on peut le faire suivre de l'indice 10 pour préciser le système employé), a pour valeur :

$$\begin{aligned} 1789_{10} &= 1\times10^{3} + 7\times10^{2} + 8\times10^{1} + 9\times10^{0} \\ &= 1\times1000 + 7\times100 + 8\times10 + 9\times1 \\ &= 1000 + 700 + 80 + 9 \end{aligned}$$

Chacun des chiffres multiplie le facteur 10 élevé à une puissance égale à son rang (compté à partir de la droite en commençant par zéro). Il en est de même avec tout autre système de numération positionnelle.

2 - LE SYSTEME BINAIRE

Dans le système binaire, on n'a droit qu'à deux chiffres (0 et 1) et tout nombre s'écrira comme une suite de 0 ou de 1, qu'il faudra additionner entre eux après les avoir multipliés, à partir de la droite, par 1, 2, 4, 8, 16, 32, 64, ... c'est-à-dire par les puissances successives de 2. Voici un exemple :

$$(111\ 1100\ 0101)_{2} = (1989)_{10}$$

$$\begin{aligned} &= 1\times2^{10} + 1\times2^{9} + 1\times2^{8} + 1\times2^{7} + 1\times2^{6} + 0\times2^{5} + 0\times2^{4} + 0\times2^{3} + 1\times2^{2} + 0\times2^{1} + 1\times2^{0} \\ &= 1024 + 512 + 256 + 128 + 64 + 0 + 0 + 0 + 4 + 0 + 1 \end{aligned}$$

En informatique, on *doit* apprendre à compter dans ce système, dès que l'on cherche à savoir avec précision ce que font les machines, car c'est le langage qu'elles utilisent. Mais les nombres binaires sont incommodes, difficiles à écrire et à retenir, car très longs. Pour ces raisons (parmi d'autres), on emploie plus souvent d'autres bases, donnant des expressions plus commodes, sans pourtant être aussi éloignées du langage des machines que l'est le système décimal.

3 - SYSTEMES OCTAL ET HEXADECIMAL

Ces deux systèmes, très employés en informatique (surtout maintenant le second) (1), dérivent du binaire. Ils mettent en œuvre les puissances successives de 8 ou de 16, c'est-à-dire :

1, **8**, **64**, **512**, **4096**...	de rang 0, 1, 2, 3, 4...	pour l'octal (base 8)
1, **16**, **256**, **4096**, **65536**...	de rang 0, 1, 2, 3, 4...	pour l'hexadécimal (base 16)

Un informaticien *doit* savoir ces séries par cœur.

Exprimons le nombre 1989_{10} dans ces deux systèmes. Pour cela, il faut diviser 1989, puis les restes, par les puissances successives de 8 (ou de 16), en commençant par la plus grande possible.

En octal :

1989, puisqu'il n'est divisible ni par 4096, ni par l'un de ses multiples, ne comporte pas plus de 4 chiffres octaux (le premier diviseur, $512 = 8^3$, est de rang 3) :

1989/512 = 3,	reste 453,	le 4e chiffre est 3
453/64 = 7,	reste 5,	le 3e chiffre est 7
5/8 = 0,	reste 5,	le 2e chiffre est 0,
5/1 = 5,	reste 0,	le 1er chiffre est 5.

Le nombre $(1989)_{10}$ s'écrit donc en octal $(3705)_8$. Vérifions sa valeur en procédant en sens inverse :

$$\begin{aligned}(3705)_8 &= 3\times8^3 + 7\times8^2 + 0\times8^1 + 5\times8^0\\ &= 3\times512 + 7\times64 + 0\times8 + 5\times1\\ &= 1536 + 448 + 0 + 5\\ &= (1989)_{10}\end{aligned}$$

En hexadécimal :

En cherchant à diviser 1989 par les puissances successives de 16 (... 4096, 256, 16, 1), on trouve les diviseurs 0, 7, 12, 5. Or on a dit qu'un système de base *N* comportait *N* chiffres ou symboles différents. Nos 10 chiffres arabes ne suffisent plus. On convient d'utiliser les 6 premières lettres de l'alphabet pour désigner les 6 chiffres supplémentaires, de valeur 10 à 15 comprises,

c'est-à-dire :	**A**	**B**	**C**	**D**	**E**	**F**
de valeur décimale :	**10**	**11**	**12**	**13**	**14**	**15**

D'où : $(1989)_{10} = (7C5)_{16}$

On vérifie : $7C5_{16} = 7\times16^2 + 12\times16^1 + 5\times16^0 = 7\times256 + 12\times16 + 5 = 1989_{10}$

Les nombres 8 et 16 étant des puissances de 2, on passe facilement des nombres octaux ou hexadécimaux aux nombres binaires et vice versa. Il suffit de grouper les chiffres binaires (on va désormais les appeler les bits) par tranches de 3 pour comparer ce nombre à l'octal, ou par tranches de 4 pour le comparer à l'hexadécimal (2). Exemple :

$(1989)_{10}$ soit :	=	0 1 1 1 0+4+2+1	1 1 0 0 8+4+0+0	0 1 0 1 0+4+0+1		(tranches de 4 bits)
	=	7	12	5		$= 7C5_{16}$
$(1989)_{10}$ soit :	=	0 1 1 0+2+1	1 1 1 4+2+1	0 0 0 0+0+0	1 0 1 4+0+1	(tranches de 3 bits)
	=	3	7	0	5	$= 3705_8$

(1) Depuis que l'*octet* est devenu la cellule mémoire élémentaire.
(2) Noter que l'écriture des nombres devient plus concise à mesure que la valeur *b* de la base croît. On montre que le nombre *n* de chiffres nécessaires pour écrire *N* est $n = 1 +$ partie entière de $(\log_b N)$.

4 - OPERATIONS EN OCTAL ET HEXADECIMAL

Il faut savoir effectuer additions et soustractions directement dans ces bases. On procède comme en base 10, colonne par colonne, à partir de la droite. On additionne les 2 chiffres ; si le résultat atteint ou dépasse la valeur de la base (8 ou 16), on lui ôte cette valeur et on *retient* 1 pour la colonne suivante. Exemples :

$4 + 3$	$= 7$	(vrai dans les bases 8, 10, 16, ...)
$4_8 + 4_8$	$= 10_8$	(on est en base 8, résultat 0 et on retient 1)
$7_8 + 6_8$	$= 15_8$	(on est en base 8, résultat 5 et on retient 1)
$7_{16} + 6_{16}$	$= D$	(en base 16, D est le treizième symbole)
$9_{16} + B$	$= 14_{16}$	(en base 16, résultat 4 et retenue 1)
$C + F$	$= 1B$	(" " , résultat B , retenue 1)
$9FFF + 1$	$= A000$	($F + 1 = 10_{16}$ et $9_{16} + 1 = A$)

La soustraction obéit à des règles similaires à celles que nous avons apprises à l'école : quand le chiffre à soustraire est supérieur au chiffre opéré, il faut ajouter à ce dernier la valeur de la base pour pouvoir effectuer la soustraction. Mais bien sûr, on devra augmenter d'une unité le chiffre à soustraire dans la colonne suivante. Exemples :

$$F - A = 5$$
$$65_{16} - 39_{16} = 2C$$

On a ajouté 16 à 5_{16}, on obtient 21_{10} avant de lui ôter 9, ce qui donne 12_{10} ($= C$), puis on ajoute la retenue 1 au 3 de 39_{16} sur la 2e colonne.

On aura remarqué qu'on omet parfois la valeur de la base en indice, soit quand le nombre est décimal, soit quand il n'y a pas ambiguïté : par exemple, la présence des lettres A...F, indique clairement qu'on utilise la base 16. Une autre façon de caractériser cette même base consiste à accoler au nombre en question la lettre H (pour *hexadécimal*).

Multiplication et division octales ou hexadécimales exigeraient de connaître par cœur leur table de multiplication, ce qui implique un effort hors de proportion avec l'intérêt présenté. Aussi, ces opérations sont-elles rarement pratiquées. Il faut surtout savoir que la multiplication d'un nombre par sa base (i.e. par un nombre de même valeur qu'elle) revient à ajouter un zéro à sa droite. Par contre, on l'a déjà dit, il est courant en informatique d'avoir à pratiquer des additions et des soustractions hexadécimales, soit pour vérifier la justesse d'un algorithme, soit pour calculer des adresses-mémoire.

5 - LOGIQUE BINAIRE

Le calcul binaire a été exposé par **Boole** en 1847 non pas pour compter autrement, mais pour représenter les règles du **raisonnement** humain. Beaucoup de propositions dans la vie courante n'ont que deux possibilités : être *vraie* ou *fausse*. Elles peuvent être liées entre elles par des conditions pour aboutir à des conséquences ou des décisions. Les mécanismes de ces raisonnements ont été analysés par les philosophes, mais Boole montra qu'ils obéissaient à des règles suffisamment précises pour bénéficier d'un traitement mathématique.

L'étude de cette science nous entraînerait trop loin et ne se justifierait que pour un informaticien se consacrant aux systèmes experts ou à l'intelligence artificielle, secteurs parmi les plus nobles de l'informatique. Pour se limiter à une initiation, on n'exposera que les quatre opérations fondamentales de la logique : la réunion, l'intersection, la réunion exclusive et l'inversion.

Les deux seules valeurs, *vrai* ou *faux*, que, par convention, peuvent admettre les propositions logiques, sont souvent confondues avec les valeurs 1 et 0 de l'arithmétique binaire. Dans ce manuel, nous assimilerons 1 à *vrai* d'une part et 0 à *faux* d'autre part (cf. chap. IX, § 7, note 11, page 78).

a - La réunion

Soit deux propositions *a* et *b*, vraies ou fausses. On appelle réunion de *a* et *b*, la proposition *r*, vraie si *a* **ou** *b* sont vraies, et fausse seulement si *a* **et** *b* sont toutes deux fausses. L'opération réunion est symbolisée par l'un des opérateurs suivants : **OU**, $|$, $\cup$, $+$. On peut écrire :

$r = a$ **OU** b $\qquad r = a \cup b \qquad r = a \mid b$ ou encore $r = a + b$

b - La réunion exclusive

La réunion exclusive (opérateur OU EXCLUSIF, **XOU**, ou $\wedge$) est telle que son résultat n'est vrai que si un seul des deux opérandes est vrai. Dans le langage ordinaire, elle est mal distinguée de la précédente (parfois appuyée par l'écriture ET/OU). La réunion exclusive se note en logique :

$$r = a \ \textbf{XOU} \ b \qquad \text{ou bien} \qquad r = a \wedge b$$

c - L'intersection

On appelle intersection de *a* et de *b*, la proposition *x* vraie seulement si *a* et *b* sont toutes deux vraies. Son opérateur est symbolisé par **&**, **ET**, $\cap$, ou même par $\times$. On rencontre donc les expressions suivantes :

$$x = a \ \& \ b \qquad x = a \cap b \qquad x = a \ \textbf{ET} \ b \qquad x = a \times b$$

d - L'inversion

Les trois opérations précédentes sont de type *binaire*, car elles lient deux propositions et, de proche en proche, un nombre quelconque de celles-ci. L'opération **inversion**, *unaire*, ne porte que sur une seule. La proposition *c* est inverse de *a* si elle est fausse quand *a* est vraie, et vraie quand *a* est fausse.

L'inversion se note en surlignant la proposition inversée *a* (on dit aussi que *c* est le complément de *a)*.

$$c = \overline{a}$$

Le résultat inverse des opérations **ET**, **OU** s'écrit par raccourci comme le résultat de l'application aux variables de l'opérateur inverse :

si $r = a \ \textbf{ET} \ b$ et $c = \overline{r}$, $\qquad c = \overline{a \ \textbf{ET} \ b} \qquad$ ou bien $\qquad c = a \ \overline{\textbf{ET}} \ b$

e - Tables de vérité

On schématise ces opérations par quatre tableaux à deux entrées, appelés tables de vérité, qui, pour tout couple d'opérandes *a* et *b* (portés sur la ligne du haut et dans la colonne de gauche), donnent les valeurs résultant de l'opération entre eux (sauf pour l'inversion, puisqu'elle n'a qu'un opérande).

OU

a,b	0	1
0	0	1
1	1	1

ET

a,b	0	1
0	0	0
1	0	1

INV

a	0	1
c	1	0

OU EXCLUSIF

a,b	0	1
0	0	1
1	1	0

En anglais, les opérateurs **OU**, $\overline{\textbf{OU}}$, **ET**, $\overline{\textbf{ET}}$, **XOU** s'appellent OR, NOR, AND, NAND, XOR, et l'inverse $\overline{a}$ de *a* s'écrit souvent NOT*a* .

f - Théorème de De Morgan

De tous les développements du calcul logique, nous ne citerons qu'une conséquence, à cause de l'importance extrême qu'elle revêt en informatique (tant logicielle que matérielle) : c'est le théorème de De Morgan. Il s'énonce :

L'inverse d'une réunion est égal à l'intersection de ses inverses.

L'inverse d'une intersection est égal à la réunion de ses inverses.

c'est-à-dire :

$$a \ \overline{\textbf{OU}} \ b = \overline{a} \ \textbf{ET} \ \overline{b} \qquad a \ \overline{\textbf{ET}} \ b = \overline{a} \ \textbf{OU} \ \overline{b}$$

Ce théorème pourrait se démontrer, en particulier à l'aide des tables de vérité. Nous ne le ferons pas et nous nous bornerons à remarquer qu'on trouve facilement dans le langage quotidien des phrases illustrant le théorème de De Morgan.

Voici par exemple deux phrases logiquement équivalentes, puisqu'elles expriment les mêmes faits et les mêmes conclusions, malgré des propositions inverses de part et d'autre :

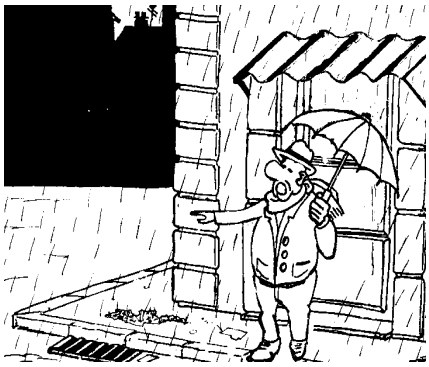
Demain, je resterai chez moi s'il pleut OU si je suis fatigué
Demain, je sortirai s'il ne pleut pas ET si je vais bien.

Dans le même ordre d'idées, on vérifiera aisément la règle ci-après, souvent utile :

L'inversion des variables dans l'opération OU EXCLUSIF n'en change pas le résultat.

$$\overline{a} \ \textbf{XOU} \ \overline{b} = a \ \textbf{XOU} \ b$$

Moi je dis qu'il faut être malade pour sortir par un temps pareil ! **(3)**



Le spécialiste en automatismes (en particulier celui agençant des systèmes de sécurité électroniques), tout comme le programmeur, doit constamment jongler avec les **OU**, les **ET** et les inversions, car il est souvent plus facile et parfois nécessaire de passer par les propositions inverses à l'aide des théorèmes ci-dessus plutôt que d'exprimer avec les propositions directes les conditions recherchées. De même, l'utilisateur des SGBD (chap. XII) doit connaître un minimum de logique.

6 - OPERATIONS ARITHMETIQUES BINAIRES

La logique gouverne tout le fonctionnement des calculateurs numériques. Nous allons montrer que même les opérations arithmétiques binaires se déduisent des opérations logiques étudiées dans la section précédente.

L'addition se pratique colonne par colonne, comme dans tous les systèmes de position. En binaire, 4 cas seulement peuvent se présenter : 0+0, 0+1, 1+0, 1+1. Ces 4 cas doivent fournir les résultats respectifs : 0, 1, 1, 10. En se reportant aux tables de vérité, on remarque qu'une addition arithmétique binaire se réduit à l'opération **OU EXCLUSIF** pour le résultat brut et à l'opération **ET** pour la retenue.

Addition binaire $a+b$: résultat = a **XOU** b
retenue = a **ET** b

En réalité, dans chaque colonne, il y a deux fois addition, puisqu'il faut ajouter à son résultat propre la retenue, égale elle aussi à 0 ou à 1, de l'opération sur la colonne précédente.

(3) La contradiction apparente entre cette boutade et les propositions logiques précédentes provient en partie du jeu sur le mot *malade*. Evidemment, un ordinateur aurait bien du mal à saisir de telles subtilités de langage !

La multiplication se déroule comme en arithmétique décimale, mais elle est bien plus simple, la table de multiplication étant des plus restreintes. Il suffit de savoir que la multiplication élémentaire d'un nombre *m* quelconque par 0 ou 1 ne peut donner que 0 ou ce nombre *m*.

Si maintenant le multiplicateur contient *n* chiffres (binaires), on effectue les additions du résultat des *n* multiplications élémentaires correspondant aux *n* chiffres du multiplicateur, après décalage d'un rang sur la gauche à chacun des chiffres. Exemple :

binaire :	*soit :*	*ou :*
1 1 0 0 1 0 1 1	CB_{16}	203_{10}
× 1 1 0 1 0	× $1A_{16}$	× 26_{10}
..... 0 0 0		
1 1 0 0 1 0 1 1 .		
1 1 0 0 1 0 1 1 . . .		
1 1 0 0 1 0 1 1		
1 0 1 0 0 1 0 0 1 1 1 1 0		
soit : 1 0100 1001 1110	= 149E	= 5278

La soustraction binaire, dans les calculateurs, se réduit à l'addition d'un nombre négatif. On parlera de ces nombres dans le chapitre VI.

La division binaire n'est guère plus compliquée que la multiplication ; elle suit les règles apprises à l'école pour la base 10, mais en bien plus simple : elle se borne à soustraire le diviseur du dividende rang par rang à partir de la gauche. Selon que la soustraction est possible ou non, on inscrit 1 ou 0 au quotient ; puis on abaisse le chiffre à droite (en fait, on se décale d'un rang) et on recommence.

Nous avons noté, à propos de la multiplication et de la division, l'intervention fréquente du *décalage*. Ce procédé est très employé en informatique. En plus de son utilisation dans les opérations précitées, le décalage d'un nombre binaire *N* de *n* rangs vers la gauche ou vers la droite, est couramment utilisé pour multiplier – ou, respectivement, diviser – *N* par 2^{n}.

On retiendra que les quatre opérations élémentaires de l'arithmétique binaire sont résolues par trois méthodes :

– l'algèbre logique,
– les décalages,
– l'utilisation des nombres négatifs.

7 - CAPACITE D'UNE REPRESENTATION NUMERIQUE

Il est indispensable de connaître le nombre de valeurs que peut prendre la représentation numérique utilisée. On sait qu'un nombre décimal de *n* chiffres peut prendre 10^{n} valeurs, par exemple 10, 100, 1000 valeurs (car le zéro est inclus) pour un nombre respectivement de 1, 2, 3 chiffres décimaux.

Il en va de même avec tout autre système de numération qui, doté d'une base *b* et de *n* chiffres, pourra représenter au maximum b^{n} valeurs. Par exemple, un nombre binaire de 8 chiffres (8 bits) peut prendre 2^{8} = 256 valeurs (de 0 à 255_{10} , ou de 0 à FF comprises), un binaire de 16 chiffres 2^{16} = 65536 valeurs, capacité égale à celle d'un nombre hexadécimal de 4 chiffres (puisque $16^{4} = 2^{16}$). On retrouve là les nombres magiques de la section 3, page 8, ces puissances successives de 16 qui jouent un si grand rôle en informatique.

EXERCICES

1 - Addition binaire

10011011	110111	11111110	1010100
+01110101	+1100011	+1011101	+10101111

2 - Soustraction binaire

10011011	10110111	11111110	11010100
-01110101	-1100011	-1011101	-10101111

Effectuez directement en binaire l'opération indiquée, puis exprimez opérandes et résultat en décimal et vérifiez la justesse du résultat par l'opération décimale.

3 - Valeurs binaires de référence

Exprimez en décimal la valeur de : 2^{10}, 2^{20}, 2^{30}.

Ces valeurs remarquables ont reçu des noms. Elles permettent de situer très vite l'ordre de grandeur d'une quantité binaire. Dites pourquoi ? (cf. chap. VI, § 4, page 44).

4 - Addition hexadécimale

Effectuer :					
	7A82	6C7D	E98C	1744D	FC0F
	+E8BA	+8F92	+9999	-F9CB	-7E81

5 - Passage octal-hexadécimal

Transcrire 7A82 puis E8BA en octal
Transcrire $75\,102_8$ puis $167\,272_8$ en hexadécimal

(développez en binaire et groupez les bits par 4 ou par 3).

6 - Tables de vérité

Dessiner la table de vérité de l'opération logique $\overline{a}$ **XOU** $\overline{b}$. Comparez-la à celle relative à a **XOU** b . Conclusion ?

Si on opère une statistique sur de nombreux couples (a,b) dont les valeurs 1 ou 0 sont distribuées au hasard, on peut dire que les opérations **ET**, **XOU**, **OU** fournissent respectivement 25%, 50% et 75% de résultats vrais. Pourquoi ?

7 - Ou exclusif

Trouvez (à l'aide des tables de vérité) la valeur remarquable de l'expression

$$z = y \text{ XOU } (y \text{ XOU } x)$$

On démontrera que z prend une valeur simple, quels que soient x et y (égaux à 1 ou 0). Cette propriété nous sera utile pour le dessin animé sur écran (chap. XIII, §2f, page 118).

Chapitre III

LES CIRCUITS ELECTRONIQUES DE BASE

1 - LES ROUAGES DECIMAUX

Les rouages des machines à calculer mécaniques utilisaient des engrenages de rapport 10. Certaines de ces roues possédaient des encoches limitant, sous la poussée de ressorts, le nombre de positions possibles à dix. En outre, la transmission du mouvement d'une roue à l'autre n'était pas continu, comme dans un engrenage classique, mais *discontinu* : on faisait avancer une roue d'un cran chaque fois que la précédente avait tourné de dix pas.

On dit que ces organes possédaient 10 *états stables*. Quand l'électronique s'est imposée à cause de sa rapidité, on a tenté de fabriquer des tubes comportant également 10 états stables. De tels tubes ont été utilisés comme *compteurs* spécialement en physique nucléaire dans les années 60 ; mais, trop difficiles à construire, ils ont été supplantés par l'électronique binaire, beaucoup plus simple.

2 - LES TUBES ELECTRONIQUES

Un tube comporte, dans une ampoule vide d'air (fig. 3-1), un filament chauffé *à blanc* qui, comme tout corps porté à haute température, émet des électrons. Sous l'effet de l'agitation thermique, chaque électron émis par le filament reçoit une certaine quantité d'énergie, variable de l'un à l'autre, mais en moyenne proportionnelle à la température. Ces électrons ne s'éloignent cependant pas beaucoup, car leur éjection du filament y crée *un trou*, c.-à-d. une charge positive, qui exerce sur eux une force d'attraction et de rappel. Si maintenant on place près du filament une électrode (une "plaque" métallique dite *anode)* portée à un potentiel positif, elle contrebalance la force de rappel pour les électrons les plus énergétiques qui sont alors *captés* par la plaque. C'est le principe de la diode, dont la propriété fondamentale est de ne laisser passer le courant *(i)* que dans un sens.

Dans la triode (fig. 3-2), on interpose une *grille* entre anode et filament et on la porte à un potentiel négatif qui contrecarre cette fois l'effet attractif de l'anode. Cette grille étant placée beaucoup plus près du filament que l'anode, son influence est beaucoup plus forte sur les électrons émis ; mais sa maille est trop lâche pour lui permettre de les capturer.

Ils sont encore collectés par l'anode. La grille, selon son potentiel, les refoule ou les laisse passer : on dit qu'elle en régule – ou *module* – le flux, c'est-à-dire le courant (toujours à sens unique).

Cette propriété permet de faire de ce tube un *amplificateur* de courant. Son invention a provoqué l'essor de toute l'électronique (radio, télévision, musique, ... pour ne citer que quelques domaines bien connus). Nous ne nous étendrons pas davantage sur les propriétés des tubes électroniques, puisque (à quelques exceptions près, dont le tube cathodique) ils ne sont plus utilisés depuis plusieurs années ni dans les calculateurs, ni dans aucun autre appareil électronique courant.

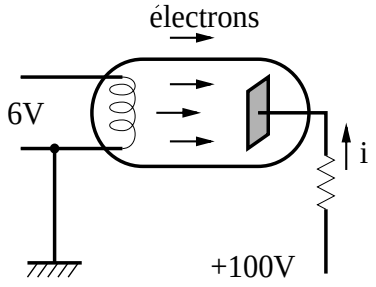


Fig. 3-1 : Diode à vide.

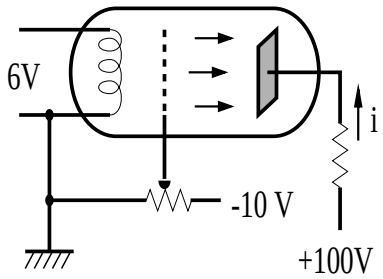


Fig. 3-2 : Tube triode.

3 - LE TRANSISTOR

Dans les solides, les atomes, situés aux nœuds d'un réseau cristallin, sont très proches les uns des autres. Certains de leurs électrons, les plus externes aux noyaux, peu liés, peuvent facilement passer d'un atome à l'autre et même, comme c'est le cas dans les bons conducteurs de l'électricité, divaguer dans tout le réseau.

Cela se produit également si l'on insère en proportion non négligeable, dans un cristal peu conducteur, une impureté "donneuse d'électrons", c'est-à-dire possédant plus d'électrons que ceux nécessaires à l'établissement des liaisons chimiques du réseau. Alors l'électron surnuméraire grossit le troupeau des électrons mobiles, à la merci du moindre champ électrique. On dit que l'on a affaire à un *semi-conducteur* de type *n*. La réciproque existe heureusement : une impureté acceptrice d'électrons, insérée dans le réseau hôte, induit une lacune, un *trou*. Ce trou pourra avoir une mobilité comparable à celle des électrons dans le matériau *n*. Ce semi-conducteur est dit de type *p*. Electrons et trous sont appelés *porteurs de charge*.

Si on met en contact intime deux semi-conducteurs (fig. 3-3a), l'un de type *n*, l'autre de type *p*, on réalise une *jonction np* qui se comporte comme une diode, parce que le courant électrique circule facilement dans un sens ("sens *direct*") et très mal dans l'autre ("sens *inverse*"), puisque les électrons excédentaires du matériau de type *n* vont chercher à combler les lacunes du matériau *p*.

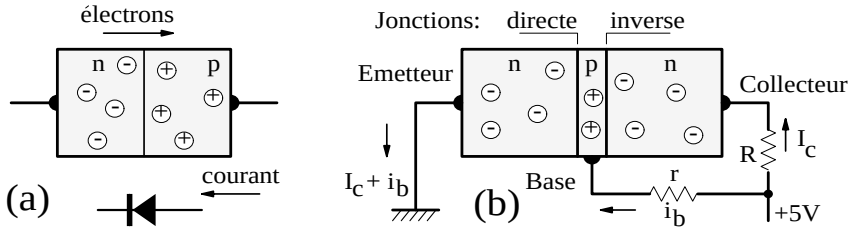


Fig. 3-3 : Diode (a) et transistor (b) par jonctions entre semi-conducteurs.

Pour fabriquer un transistor, on "empile" trois tranches alternées de ces semi-conducteurs, par exemple une tranche *p* entre deux de type *n* (le transistor est alors de type NPN). Ces tranches constitueront les trois électrodes du transistor : *émetteur, base, collecteur* (fig. 3-3b). Deux circuits électriques sont ensuite créés extérieurement : l'un, entre base et émetteur, *polarise* dans le sens direct la jonction *pn* ; il conduit alors facilement le courant, lequel ne sera limité que par la forte résistance *r* insérée dans ce circuit. L'autre circuit, entre émetteur et collecteur, englobera deux jonctions, dont l'une est *en sens inverse* : à cause de cela, il ne laisse passer normalement aucun courant.

Franchissant aisément la jonction directe, les porteurs de l'émetteur s'infiltrent dans la base. Sous certaines conditions (épaisseur de base faible, durée de vie des porteurs grande...), ils vont aller perturber la jonction opposée et, bien qu'en sens inverse, la franchir "sur leur élan" pour rejoindre le collecteur. La base, trop mince, ne les absorbe pas et n'en collecte que très peu. Elle se comporte comme la grille des triodes : son courant faible i_b *module* le fort courant I_c qui s'établit entre émetteur-collecteur.

4 - L'AMPLIFICATEUR

On peut voir dans la figure 3-4 comment on schématise et on monte un transistor : l'émetteur (repéré par une flèche en série) sera relié *à la masse* par une faible résistance r_e, le collecteur à l'alimentation via une résistance *R* de valeur moyenne. La tension *U* de cette alimentation est de quelques volts, typiquement 5 et rarement plus de 15.

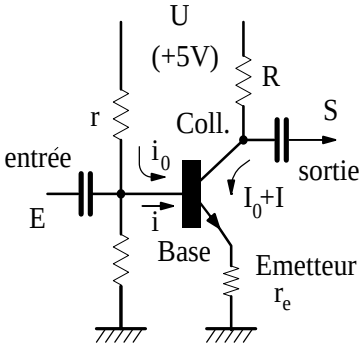


Fig. 3-4 : Transistor amplificateur.

La base est reliée à la même alimentation (ou à une autre) par une forte résistance *r*. Il passe dans ce circuit un courant faible $i_0 \approx U/r$ appelé *courant de polarisation*. C'est dans ce circuit qu'on injecte, via un transformateur ou bien un condensateur, le *signal d'entrée i* généré par la tension *E*.

Dans le circuit de sortie – celui du collecteur –, circule en permanence le courant de repos I_0 sous l'influence de i_0 et, en superposition, sous l'influence de *i*, le courant $I = G\,i$ (*G* est le gain du transistor). Ces courants *I* et *i* provoquent la variation de tension *RI* ou *GRi* aux bornes de la résistance *R* du collecteur. On prélève ces variations à l'aide d'un condensateur ou d'un transformateur : ce sont elles qui constituent le signal ayant un intérêt pratique.

L'amplificateur est extrêmement employé dans les appareils modernes (et pas seulement dans les *chaînes* musicales). Un amplificateur usuel comprend toujours plusieurs transistors, soit pour en augmenter le *gain* (celui d'un transistor unique ne dépasse guère 20 et beaucoup moins en pratique), soit pour améliorer sa *fidélité* (non-déformation du signal), soit pour le rendre différentiel, c-à-d. capable d'amplifier non plus une tension E mesurée par rapport *à la masse*, mais la différence de tension $E_+ - E_-$ entre deux conducteurs. Le montage comprend deux transistors comme ci-contre, mais avec une résistance d'émetteur commune (elle assure le *couplage* entre les deux transistors). Une seule des deux sorties S est utilisée. Dans les schémas, il est représenté par le symbole ci-après :

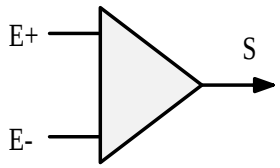


Figure 3-5 : Amplificateur différentiel schématisé.

5 - LE SIGNAL ET SA VARIATION

On appelle *signal* tout phénomène porteur d'une information. En général, il s'agit d'un courant ou d'une tension électrique. Quel qu'il soit, le phénomène n'apporte d'information que s'il varie dans le temps **(1)**.

La variation – ou *dérivée* – d'un signal est elle-même un signal, qui se déduit assez facilement du premier :

– elle est égale à zéro si le signal est constant,

– elle est positive si le signal croît, négative s'il décroît,

– son amplitude croît avec le taux de variation du signal.

La variation d'un signal électrique s'extrait par des *composants* simples : un condensateur en dérivation ou un transformateur en série réalise cette fonction sous des conditions généralement faciles à respecter (qui déterminent la *bande passante*).

Les signaux habituels en électronique "classique" (par exemple dans la radio, la musique ...) varient *sinusoïdalement* dans le temps : on les représente par des courbes ayant l'allure de celle de la figure 3-6*a*. Leur dérivée a même allure (avec cependant un décalage). Les signaux de l'informatique, eux, ont un comportement temporel voisin de la courbe *b* de cette même figure ; une telle courbe pourrait représenter par exemple la tension d'une ligne télégraphique sur laquelle un préposé envoie un message codé en *morse* (un tel signal est qualifié de rectangulaire). Sa dérivée est représentée par la courbe *c* de cette figure. On vérifiera que cette dérivée répond bien aux caractéristiques énoncées ci-dessus, étant formée d'impulsions brèves, positives ou négatives selon le sens de la variation. Ce comportement est celui de la dérivée de tous les signaux de type rectangulaire, puisqu'ils varient brusquement.

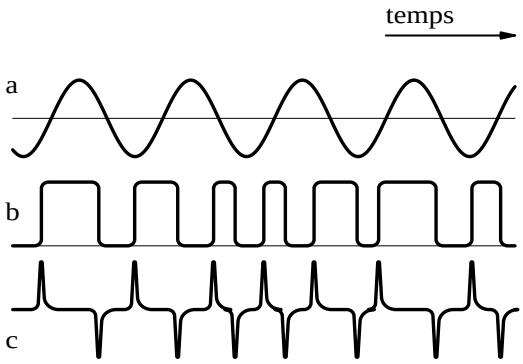


Fig. 3-6 : Signaux électroniques (a : sinusoïdal ; b : rectangulaire ; c : impulsionnel).

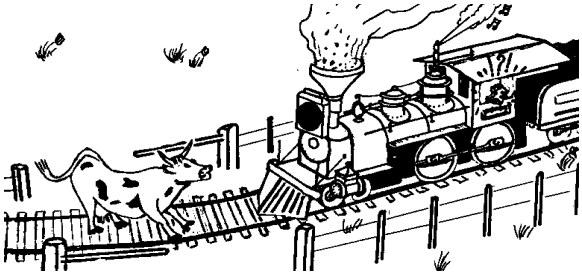
(1) Par exemple un son de sirène continu n'apporte que deux informations : l'une à sa mise en route (présence), l'autre à son arrêt (durée). Ce sont ses deux seules variations. Mais on peut donner plus d'information en *modulant* la sirène (autre variation).

6 - LE TRANSISTOR INVERSEUR

En informatique, les transistors sont très peu employés sous la forme décrite dans la section 4 qui concerne plutôt les amplificateurs de type *proportionnel*. On les fait toujours fonctionner en *tout ou rien*, **comme un interrupteur**, parce que ce régime est très sûr et très peu sensible aux signaux parasites **(2)**.

Il n'y a pas de demi-mesure : *il faut qu'une porte soit ouverte ou fermée*, le courant doit être maximum ou nul. Les transistors des circuits informatiques ne connaissent que deux états, de symbole 1 et 0 ; on étudie leur fonctionnement grâce à l'arithmétique binaire.

L'amplificateur de la figure 3-4, tout en restant *câblé* de manière similaire, dégénère alors en un circuit appelé inverseur. Le signal *E*, maintenant appliqué directement (sans condensateur) à l'entrée, ne peut être que nul ou non nul, c'est-à-dire dans ce dernier cas, suffisant pour saturer le transistor, lui faire débiter son courant maximum, légèrement inférieur à *U/R*. Si E est nul, le courant *i* est nul et le potentiel de sortie *S* au pied de la résistance *R* (ou au collecteur) est *U* (5 volts par exemple). Si E n'est pas nul, le courant *i* est maximum et S voisin de 0. On reconnaît ici les règles de l'inversion logique : $0 \rightarrow 1$ et $1 \rightarrow 0$. D'où le nom de ce montage, le plus employé de toute l'informatique.



Il faut qu'une porte soit ouverte ou fermée.

7 - LA BASCULE

La bascule est également un circuit fondamental, imaginé d'abord pour compter des événements aléatoires. Elle est formée de deux inverseurs fortement couplés, la sortie de chacun étant injectée à l'entrée de l'autre (fig. 3-8). Dans un inverseur, la variation du signal de sortie *S* est toujours opposée à celle du signal d'entrée *E* ou *i*. Si l'on monte deux inverseurs en cascade, avec l'entrée *E2* du second reliée à la sortie *S1* du premier, les deux inversions se compensent (on dit que *E1* et *S2* sont en phase). Pourtant ces transistors amplifient beaucoup : un tout petit signal sur *E1* provoque un fort signal *S2* de même sens. Si alors, on injecte *S2* sur l'entrée *E1*, que va-t-il se produire ?

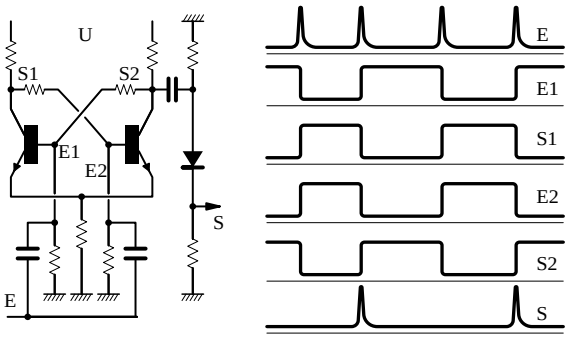


Fig.3-8: Bascule électronique: schéma et fonctionnement **(3)**.

Eh bien, on aura affaire à un système en réaction positive, dans lequel la sortie renforce considérablement l'entrée : il amplifie tellement que le premier transistor atteint la saturation dès le début du signal *E1* et que le second "se coupe" aussitôt. Ainsi, un tout petit signal, une petite impulsion sur *E1*, fait *basculer* tout le montage. Mais le dispositif, bien qu'insensible alors à toute nouvelle sollicitation sur *E1*, reste sensible à un signal sur *E2* : le circuit rebascule alors de l'autre côté. D'où son nom, bien compréhensible.

Les impulsions d'entrée peuvent être envoyées à la fois sur *E1* et sur *E2*, une seule entrée étant sensible à un instant donné. A chaque impulsion, le système bascule, une fois d'un côté, une fois de l'autre, selon la courbe *S1* ou *S2*, fig. 3-8. On dit que les sorties *S* passent alternativement par les états haut et bas. Si maintenant on dérive le signal de sortie *S2*, on obtient des impulsions brèves alternativement positives et négatives (comme dans la courbe *c* de la figure 3-6). On supprime alors celles d'une même polarité, par exemple les négatives, au moyen d'une diode : il reste le signal de la courbe *S*. La bascule, alimentée en impulsions, fournit encore des impulsions, *mais seulement à raison d'une sur deux*.

Ce circuit est parfaitement symétrique, excepté sa sortie, prélevée d'un seul côté. On peut rendre visible son fonctionnement en branchant une petite lampe (on dit "un voyant lumineux") entre le point *S2* et la masse. La lampe s'éteint ou s'allume alternativement à chaque impulsion. Un dispositif électro-mécanique voisin peut aider à comprendre ce comportement : le

(2) Il convient de mentionner ici le calcul analogique, utilisant les propriétés d'amplificateurs spéciaux *(opérationnels)* dérivés du type proportionnel. Bien adapté à la résolution d'équations différentielles, ce mode de calcul a malgré tout été supplanté vers 1970 par le calcul numérique (beaucoup moins affecté par les bruits parasites).

(3) C'est sur un *oscilloscope*, appareil de mesure comportant un écran cathodique, qu'on peut voir ces signaux comme ils sont représentés ici.

"télérupteur" que les électriciens installent dans nos habitations pour actionner l'éclairage. Chaque pression sur le bouton-poussoir fait changer l'état de l'éclairage : une fois, on allume, une fois, on éteint, comme dans une bascule.

8 - LE COMPTEUR

On place maintenant *N* bascules les unes à la suite des autres en reliant la sortie *S* de chacune à l'entrée *E* de la suivante. Chacune ne transmet à sa voisine qu'une impulsion sur deux. Ce dispositif, appelé *compteur*, *registre* ou *échelle*, est capable de compter jusqu'à 2^{N} impulsions. Pourquoi ? Parce que, si un tel montage a reçu au total *m* impulsions *($m < 2^{N}$)*, la suite de ses bascules adopte une configuration qui nous permet de connaître *m* sans aucune ambiguïté.

Cette propriété peut se démontrer, mais on l'admettra intuitivement en examinant bien la fig. 3-9. On y représente le fonctionnement de *N*=4 bascules au fur et à mesure de l'arrivée d'impulsions. Après réception d'un nombre pair d'impulsions, toute bascule revient à son état initial et envoie une impulsion à la bascule suivante pour la faire changer d'état. Le compte des impulsions se propage ainsi d'une bascule à l'autre. En bas de la figure 3-9, on relève l'état de toutes les bascules après chaque impulsion sur l'entrée générale, en attribuant le *bit* 1 aux bascules dans l'état haut et le *bit* 0 à celles dans l'état bas. Si l'on avait branché des voyants sur ces bascules, les 0 et les 1 correspondraient respectivement à des voyants éteints et allumés.

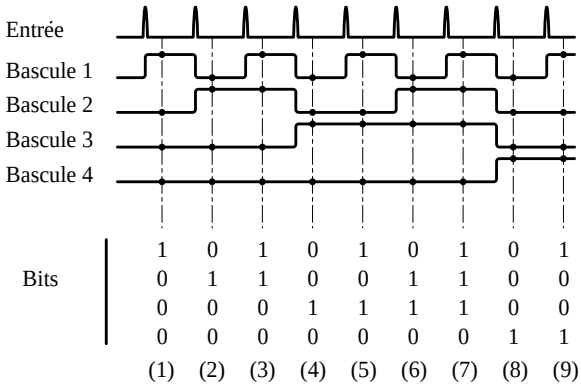


Fig. 3-9 : Compteur électronique.

Ecrivons maintenant les bits de gauche à droite, plutôt que de bas en haut : ils vont former les nombres de *N* (c.-à-d. ici 4) chiffres suivants :

0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, ..

On reconnaît dans cette suite les nombres 1, 2, 3, 4, 5, 6, 7, 8, 9, ... écrits en binaire. La configuration du registre représente à tout instant en binaire le nombre d'impulsions reçues.

9 - LES CIRCUITS LOGIQUES

D'autres circuits, très importants pour l'informatique, se réalisent également à l'aide de diodes et de transistors. Examinons le montage de la figure 3-10, dans lequel deux diodes amènent les tensions *a* et *b* sur la résistance *R*. Si l'une ou l'autre entrée, ou les deux, sont à 1 (c'est-à-dire à 5 volts), la borne *S* de *R* sera aussi à 1. Pour que *S* soit à zéro, il faut qu'à la fois *a* et *b* soient nulles. Ce montage fonctionne donc selon la table de vérité de l'opérateur OU et on l'appelle **circuit OU** (les diodes ont pour rôle pratique de rendre les entrées indépendantes, aucun courant ne pouvant passer de *a* à *b*, ni de *b* à *a*).

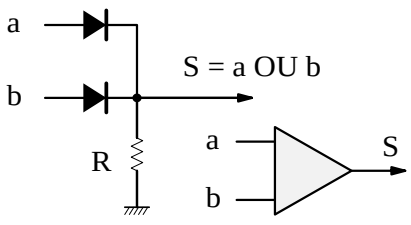


Fig. 3-10 : Circuit OU *à diodes.*

Dans la figure 3-11, deux diodes d'entrée attaquent la base d'un inverseur. Quand l'une *ou* l'autre entrée *a* ou *b* est à 0, le courant de base dû à *r* est court-circuité. Le transistor ne conduit plus et *S monte* à 1. Quand les deux entrées sont à 1, le transistor, alimenté par *r*, conduit et met à 0 le potentiel de la sortie *S*. C'est donc un circuit $\overline{\text{ET}}$ (NAND), comme le prouve la table de vérité du $\overline{\text{ET}}$.

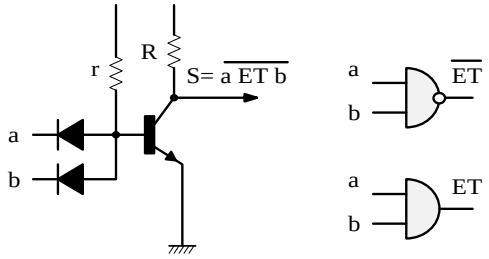


Fig.3-11 : Circuit $\overline{\text{ET}}$ *ou* NAND *à transistor*.

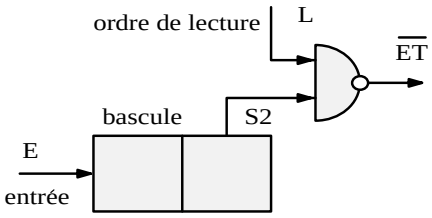
Les formes en grisé dans les figures ci-dessus sont les symboles des circuits logiques OU, ET et $\overline{\text{ET}}$, lorsqu'on ne veut pas détailler la façon dont ils sont réalisés. Remarquer le petit cercle qui signale l'inversion.

Le circuit $\overline{ET}$, associé à des inverseurs, permet de réaliser, en exploitant le théorème de De Morgan, **n'importe quel opérateur logique**.

Selon le chapitre II §6 (p. 11), les opérations arithmétiques binaires dérivent des opérations logiques. Les circuits qu'on vient de décrire sont donc capables d'**assurer toutes les fonctions arithmétiques** d'un calculateur numérique binaire.

10 - LA FONCTION MEMOIRE

Le calcul requiert une autre fonction importante, celle de mémoire. La bascule de la page 18 est capable de jouer ce rôle, puisqu'après avoir basculé d'un côté ou de l'autre, elle conserve cet état. C'est une cellule-mémoire élémentaire pouvant mémoriser 1 *bit* (une unité d'information) presque indéfiniment, en fait jusqu'à coupure des alimentations ou à réception par *E1* ou *E2* d'une nouvelle impulsion. On peut lire le contenu de cette cellule en lui adjoignant une "porte" de lecture *L*, c.-à-d. un circuit ET dont une entrée est reliée à la sortie *S2* de la bascule et l'autre au conducteur acheminant les ordres de lecture (fig. 3-12).



F*ig.3-12 : Cellule-mémoire à bascule.*

Ce type de mémoire s'appelle **mémoire vive** ou **RAM** *(random access memory)*, mémoire d'*accès aléatoire*. C'est un circuit dans lequel on peut aller lire et écrire *directement* (au lieu d'*aléatoire*, il vaudrait mieux dire *direct*, cf. note 7, page 22).

La bascule *(flip-flop)* est effectivement employée pour fabriquer les mémoires vives les plus exigeantes. Cependant, on a mis au point un **autre type de mémoire** comportant moins de transistors et donc moins cher. C'est un condensateur qui en constitue l'organe fondamental de rétention **(4)**. Toutefois ce composant, se déchargeant assez vite, doit être *régénéré* périodiquement (10 à 100 fois par seconde). Ces mémoires sont appelées pour cela à *rafraîchissement* ou **dynamiques**, en anglais DRAM. Par opposition, la mémoire à bascule est appelée SRAM (S pour "statique"). La constitution de certaines DRAM est très proche de celle des récepteurs CCD qui équipent maintenant les caméras vidéo. Elles sont moins rapides que les SRAM.

Ces cellules-mémoire sont intégrées dans une puce, qui en contient des dizaines ou des centaines de milliers, selon l'avancement des techniques employées en micro-électronique (actuellement, on en est au stade **VLSI**, *very large scale integration*, avec des transistors de la dimension du micron).

11 - LES MEMOIRES MORTES

Tous les circuits-mémoire ci-dessus ont l'inconvénient de perdre leurs données quand l'alimentation électrique est *coupée*. On a pourtant souvent besoin de conservation pendant une durée beaucoup plus longue. Deux types de cellule-mémoire permanente ont été développés dans cette intention : l'un repose sur le **magnétisme** (description dans la section 12), l'autre recouvre la notion de **ROM** *(read only memory)*. ROM est souvent opposée à RAM, ce qui est illogique, car la ROM fait bel et bien partie des mémoires "d'accès aléatoire". Les appellations *mémoire vive* et *mémoire morte* seraient beaucoup plus rationnelles.

On a imaginé différents artifices pour créer une cellule élémentaire de mémoire morte. Le plus ancien consiste à graver, sur un *substrat* isolant, deux réseaux ou nappes de conducteurs électriques isolés entre eux, du moins à l'origine. On relie ensuite par des diodes certains conducteurs entre l'un et l'autre *réseau*. Cette connexion s'opère par évaporation sous *vide* d'un semi-conducteur au travers d'un masque. Ce procédé lourd ne se justifie que pour des séries importantes. Le nom de **ROM** paraît maintenant réservé à ce type de produit.

Un peu plus souple est le procédé consistant à *claquer* (fondre) de *minuscules fusibles* dans un réseau. Ce type de mémoire morte est appelé programmable ("**PROM**") : son possesseur peut en effet, grâce à un appareil spécial, le *programmateur*, écrire ses données ou son programme dans une telle puce, mais une seule fois.

D'autres types de mémoires mortes reposent sur un principe voisin de celui des DRAM. Un condensateur totalement isolé au sein du matériau retient l'information et influence l'électrode de commande d'un

(4) Ce condensateur est relié à l'électrode de commande d'un transistor qui est ici du type "à effet de champ". Dans un tel transistor, très utilisé en informatique, le courant principal est commandé, comme dans un tube, par la *tension* de l'électrode-*porte* et non, comme dans un transistor classique, par le *courant* de la base. Le courant d'entrée est alors extrêmement faible, ce qui est un très gros avantage.

transistor à effet de champ. Le temps de conservation devient très long (plusieurs années). Il est de plus possible d'effacer de telles mémoires et de les *reprogrammer*, mais avec des moyens non inclus dans le calculateur. Ces mémoires inaccessibles en écriture pour l'ordinateur et gardant leur information, même alimentation coupée, font encore partie des ROM.

On les appelle **EPROM** (ROM programmables et effaçables). Avant de les réutiliser en écriture, on peut les effacer selon deux procédés : l'un consiste à irradier les condensateurs de la puce par un *rayonnement* ultra-violet intense (elles s'appellent alors **REPROM**, R signifiant rayonnement, et comportent une fenêtre de quartz, ce qui les rend chères). Le second permet de *briser* momentanément l'*isolant* du condensateur à l'aide d'une tension électrique beaucoup plus élevée que celles du fonctionnement normal : ce sont les **EEPROM** (effaçables électriquement, appelées aussi mémoires-*flash*). Ce sont certainement elles qui connaîtront à l'avenir le plus fort développement : elles pourraient même remplacer les disques.

12 - MEMOIRES MAGNETIQUES

On sait qu'on peut aimanter un barreau d'acier à l'aide du champ magnétique produit par un courant électrique dans un bobinage entourant le barreau. On sait de même qu'il existe deux sens d'aimantation (appelés nord et sud).

Quand un barreau a été aimanté, il est difficile de changer le sens de son *aimantation*. Ceci provient de la résistance au changement qu'opposent les électrons responsables du magnétisme dans les atomes du matériau. Lorsqu'il atteint une valeur suffisamment élevée, appelée *champ coercitif*, le champ extérieur parvient à vaincre cette inertie et fait basculer l'aimantation dans le sens imposé, orientation que l'aimant gardera jusqu'à l'application d'un nouveau champ d'intensité au moins égale à celle du champ coercitif.

Un barreau d'acier garde donc la *mémoire* du dernier champ suffisamment intense qui lui a été appliqué (phénomène dit de *rémanence)*. La figure 3-13 représente la variation de *l'aimantation interne I* du barreau en fonction du *champ extérieur H* **(5)** .

On note, sur la droite de la figure, que le champ *H* a été suffisamment fort pour amener l'aimantation *I* à son niveau le plus élevé. L'aimantation conserve ce niveau lorsque le champ responsable de celle-ci a été supprimé (c.-à-d. ramené à la valeur H=0).

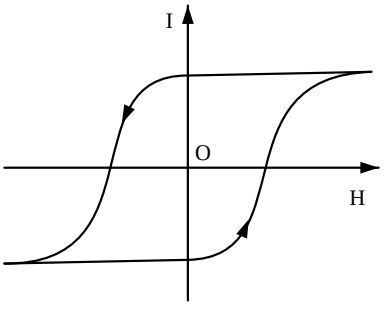


Fig. 3-13 : Hystérésis magnétique.

On retrouve le même comportement – mais en plus net encore – quand on remplace le barreau par de petits anneaux (les **tores** à mémoire des années 60), ou même par de la poudre formée de grains magnétiques microscopiques de *ferrite* collés sur un ruban plastique. Chaque grain, entraîné par le ruban, passe à un moment donné sous la *tête enregistreuse* et conservera la mémoire du champ qu'elle produisait à ce moment-là.

Différents codes sont employés pour représenter les valeurs binaires 1 et 0. Par exemple, on peut convenir de leur faire correspondre des aimantations de sens différents. Dans la figure 3-14, on peut voir le schéma d'une tête enregistreuse de ce type. C'est un minuscule électro-aimant. La tête de lecture aurait la même allure, mais elle détecterait les changements d'aimantation et non les aimantations elles-mêmes. Ces procédés sont à l'origine non seulement de l'**enregistrement magnétique** pour l'informatique, mais aussi de celui des sons sur *bande* et sur *cassette* et de celui des films en *vidéo-cassettes*.

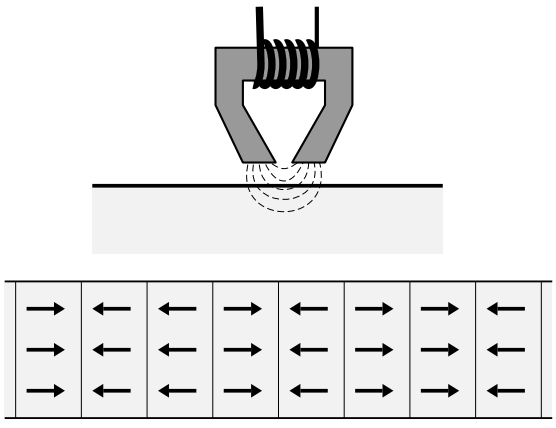


Fig. 3-14 : Tête de lecture-écriture magnétique et schéma d'une piste de disque magnétisée par la suite de bits 10010110 .

(5) Cette courbe est appelée *courbe d'hystérésis* parce que, si on fait varier continûment (entre deux valeurs de signe opposé) le champ *H* appliqué au barreau et qu'on en mesure en permanence l'*effet* – l'aimantation *I* – celle-ci apparaît comme *en retard* (= hystérésis) sur sa *cause H*.

13 - LES MEMOIRES OPTIQUES

Dans le premier de ces procédés, les informations sont conservées par un disque optique sous forme de micro-cuvettes – de la dimension du micron – gravées avec un laser, par fusion ou par brûlure, sur la surface du disque original. Deux moulages successifs permettent d'en tirer d'abord une matrice en négatif, puis des copies conformes à l'original. Ces disques sont ensuite métallisés pour les rendre réfléchissants. Tout cela ne peut être fait qu'en usine.

L'usager ne dispose que d'un **lecteur**. Selon la figure 3-15, une diode laser éclaire ponctuellement la surface du disque. Si celle-ci est plane (bit 0), elle réfléchit la lumière en un pinceau aussi fin que celui incident. Après réflexion partielle sur le *coupleur C* (lame semi-réfléchissante), ce pinceau se dirige vers le détecteur *D*, mais il est arrêté par un obturateur de la taille adéquate. Par contre, si la surface de disque présente une cuvette (bit 1), le faisceau incident sera diffusé, donc élargi ; après réflexion partielle sur le coupleur, il atteindra le détecteur puisqu'il déborde l'obturateur. On pourra rencontrer des systèmes quelque peu différents, sans obturateur par exemple, mais avec plusieurs diodes excentrées. Quoi qu'il en soit, ce procédé est le même que celui employé avec les disques *audio numériques* (appelés *compacts* ou *laser)*. D'ailleurs leurs lecteurs sont souvent capables de lire aussi bien les disques audio que ceux destinés à l'informatique.

Il reste à dire quelques mots du **disque optique réinscriptible**, c.-à-d. utilisable en *écriture* comme en lecture. Il s'agit là en réalité de mémoire magnétique, aimantée transversalement, lue et écrite par un faisceau laser. Celui étant beaucoup plus "fin" que l'espace magnétisé par un électro-aimant, la *résolution* (la finesse) sera bien meilleure (500 fois environ en surface, domaines magnétisés de la taille du micron).

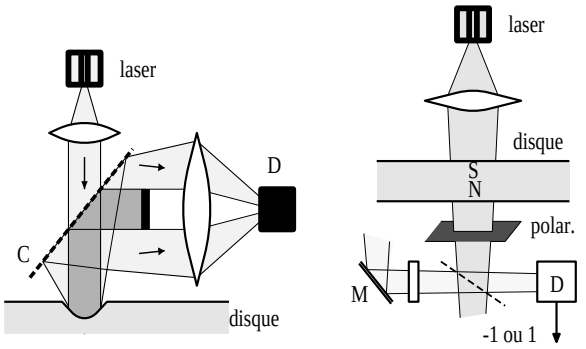


Fig. 3-15: Lectures optique et magnéto-optique.

Sur un disque au préalable magnétisé transversalement de façon uniforme, le faisceau laser produit un échauffement local. La température est suffisante (dépassement du point de Curie) pour que le domaine se désaimante ou bien se magnétise en sens inverse si on a appliqué un champ externe convenable. Après passage du faisceau, le domaine conserve sa nouvelle aimantation comme dans tout procédé magnétique.

La lecture pourrait se faire selon le schéma de droite de la figure 3-15. Le faisceau polarisé d'une diode laser traverse le domaine sondé. Il y subit l'*effet Faraday* : son vecteur-champ tourne dans le plan d'onde d'un petit angle θ donné par la loi de Verdet. Le polariseur sous le disque analyse le faisceau selon ses deux composantes : il n'en laisse passer que la fraction en $sin\,\theta$ due à l'effet Faraday et qui seule atteint le détecteur. Si le domaine sondé a été désaimanté, aucun signal n'est détecté. Cette situation peut représenter un bit 0. Il est cependant bien préférable de représenter les 1 et les 0 par des aimantations de signe contraire, auquel cas la détection est un peu plus complexe et nécessite par exemple un *homodynage* **(6)** avec une fraction de lumière prélevée sur le laser avant sondage.

Les disques-mémoire reposant sur ces principes ne sont pas encore très répandus, mais le lecteur initié à l'informatique se devra de suivre leur progression au fil des ans.

Tels sont les principes employés pour donner de la mémoire **(7)** aux ordinateurs et *conserver* l'information. On aura remarqué l'importance de la physique de *l'état solide* (semi-conducteurs) et celle du magnétisme. Sans les découvertes majeures des physiciens en ces domaines (Bardeen, Brattain, Shockley, P. Curie, Langevin, Weiss, L. Néel), l'informatique n'aurait pas connu l'essor qui est le sien.

Après avoir cherché à comprendre de quoi étaient faites les cellules élémentaires de traitement et de rétention de l'information électronique, nous verrons dans les deux prochains chapitres comment on assemble ces cellules de base pour réaliser les organes des calculateurs.

(6) L'homodynage est une technique amplifiant fortement avec peu de bruit tout en tenant compte du signe ou de la phase du signal, mais elle nécessite un signal de référence. Détection et émission peuvent être du même côté d'un disque réfléchissant, l'effet Faraday doublant par aller-retour dans le matériau.

(7) Les mémoires optiques ou magnétiques sur bandes, disques ... sont dites *d'accès séquentiel*, par opposition à celles d'*accès direct* (RAM, ROM, p. 20), car il faut en lire un grand nombre *en séquence* avant d'atteindre celles recherchées.

EXERCICES

1 - Continuez le graphe de la figure 3-9, page 19, jusqu'à la 17e impulsion. Commentez le résultat.

2 - Rappelez les différences entre RAM et ROM. Peut-on parler de mémoires à transistors ? Si oui, quels types de mémoire méritent cette appellation ? Donnez la signification des termes PROM, EPROM, REPROM, EEPROM.

3 - Imaginez un schéma électrique fait de circuits ET et OU, transformant une série de 4 bascules en un compteur BCD (c.-à-d. capable de compter jusqu'à 10) **(8)**.

4 - Le compteur décrit dans ce chapitre (§8) possède une analogie très grande avec deux – et seulement deux – des compteurs plus ou moins usuels ci-après :

– le compteur EDF domestique,
– le compteur à eau,
– le compteur de taxes téléphoniques,
– le compteur de vitesse sur le tableau de bord des voitures,
– le compteur des voitures passant sur une route (tube de caoutchouc sur la chaussée).

Dites avec lesquels il y a analogie, en réfléchissant sur le type des "objets" comptés. Dans tous les autres cas, l'appellation *compteur* est abusive.

5 - On appelle couramment *porte* un circuit logique qu'on interpose entre un émetteur et un récepteur. L'émetteur (binaire) n'envoie que des 1 ou des 0. La porte ne doit laisser passer les valeurs 1 que si elle est ouverte. Ecrivez sa table de vérité. Quel circuit élémentaire permet de la réaliser ?

(8) A la 9ème impulsion, le compteur doit être placé *de force* dans l'état qu'il aurait à la 15ème s'il comptait *en binaire pur*.

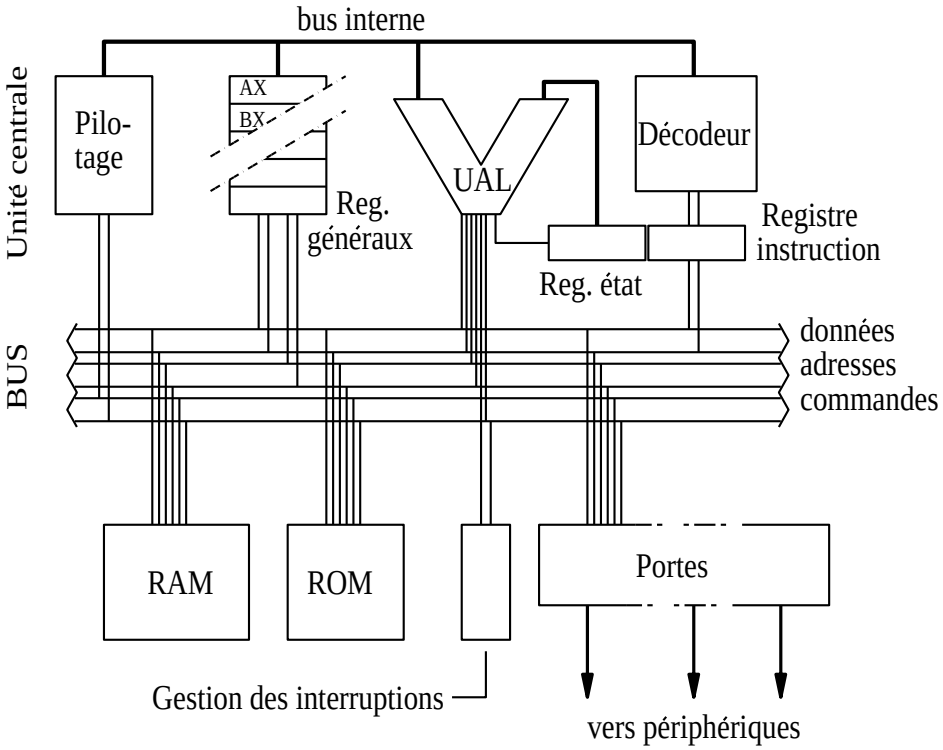


Fig. 4-1: Schéma simplifié du cœur d'un calculateur.

Chapitre IV

LE CŒUR DU CALCULATEUR

Dans une installation informatique destinée au calcul ou à la bureautique, on peut distinguer le cœur, qui sera détaillé ici, et les périphériques, qu'on décrira dans le chapitre suivant. La frontière qui les sépare n'est pas géographique (le même boîtier peut contenir le cœur et quelques périphériques), mais elle repose sur le type de communication qui s'établit entre eux : les constituants du cœur ont une homogénéité suffisante pour pouvoir dialoguer tous à la même vitesse et sous les mêmes normes.

On peut considérer que font partie du cœur l'unité centrale, la mémoire rapide (vive et morte), le canal d'échange, les portes d'entrées-sorties et les circuits d'interruption (se reporter à la figure 4-1 en face).

On appelle *carte-mère* la carte électronique la plus importante d'un calculateur de type *personnel*. Elle contient à peu près ce qu'on appelle ici le cœur de la machine, ainsi que des circuits annexes dont nous ne parlerons pas (gestion des bus en particulier).

1 - L'UNITE CENTRALE

Appelée également **micro-processeur** ou par abréviation **UC** (CPU en anglais), c'est un circuit intégré – *une puce* – de hautes performances. Malgré sa taille (quelques cm^2, voir figure ci-contre), il comporte des centaines de milliers ou des millions de transistors et assure toutes les fonctions de calcul et d'ordonnancement. On peut en détailler les sous-ensembles ci-après :

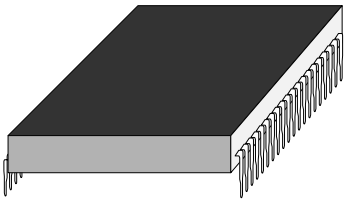


Fig. 4-2 : Un microprocesseur ou UC (4 à 5 cm de côté, au moins 40 broches).

a - L'unité de pilotage et de synchronisation

Cette unité comporte une *horloge* de cadencement, généralement peu précise, dont la fréquence va gouverner la vitesse générale de l'UC. En fait d'horloge, il s'agit plutôt d'un simple oscillateur de synchronisation. Sa fréquence peut dépasser 100 Mhz (100 millions d'impulsions par seconde) dans certains calculateurs actuels, même de type "personnel".

Jamais en repos, ce module va forcer l'UAL (unité arithmétique et logique, cf. § c) à aller chercher les instructions et à les exécuter à son rythme. C'est lui également qui assure, à la mise sous tension de l'appareil, le *branchement* de l'UAL à un endroit précis de la mémoire morte pour assurer le démarrage général *(amorçage, bootstrap)*.

b - Le décodeur d'instructions

Le *décodeur* est un module, formé de ET et de OU, qui transforme chaque instruction qu'il reçoit en un *micro-programme*, séquence de signaux électriques excitant le canal d'échange et l'UAL au rythme de l'horloge. Souvent constitué d'une ROM, c'est l'un des plus importants modules du calculateur, puisque, définissant son *jeu d'instructions*, il en conditionne toute la puissance et l'efficacité (cf. chapitre VII).

c - L'unité arithmétique et logique

L'*unité arithmétique et logique* (UAL) est le module qui effectue les calculs, les opérations binaires, les décalages ... et "prend les décisions" quand une alternative se présente.

Elle va chercher dans la mémoire à la fois des instructions et des données et traite ces dernières en fonction des instructions reçues avant de les remettre en mémoire. On appelle **donnée** toute information susceptible d'être contenue en mémoire, autre qu'une instruction ou une adresse. La donnée élémentaire manipulée par l'UAL s'appelle un **mot**.

L'UAL n'effectue en principe d'opérations directes que sur des nombres entiers (selon les idées exposées dans le second chapitre). On verra plus tard (chap. VI) que les calculateurs peuvent travailler également sur des *nombres flottants*, c.-à-d. *décimaux* : pour accélérer ce genre de calcul (qui ne se présente que si l'on exécute un programme mathématique ou graphique), on peut adjoindre à l'UAL un *coprocesseur mathématique*, spécialement conçu pour décoder et exécuter les instructions manipulant des nombres flottants. En son absence, le *compilateur* (voir page 75) devra adapter les opérations flottantes aux possibilités de l'UAL.

d - Les registres centraux

L'UAL a besoin, au cours de ses calculs ou de ses manipulations, d'avoir *à portée de main* les valeurs ou les nombres utilisés à intervalles rapprochés. Leur sauvegarde en mémoire centrale (cf. page 27) exigeant plusieurs cycles d'horloge, on gagne du temps en les rangeant temporairement dans des *registres* spéciaux à proximité de l'UAL.

On appelle **registre central** une suite de cellules-mémoire (vive) d'une taille égale à celle du *mot* normal manipulé par l'UAL et implantée tout près d'elle dans l'UC. Les registres centraux sont également appelés *registres généraux* ou *registres* tout court. Ils sont capables actuellement de manipuler 16, 32, voire 64 bits à la fois, mais on pourra les décomposer en registres plus petits (de 8, 16... bits), pour s'adapter à la taille des divers *objets* informatiques employés.

Selon leur vocation, on les répartit en trois catégories :

α - registres de travail

Ces registres, peu nombreux (une dizaine au plus) sont tous *banalisés* et peuvent *mémoriser* temporairement n'importe quelle donnée ou adresse. Cependant, certaines instructions font appel à des registres bien précis ; d'autres se déroulent plus vite si on utilise les registres *ad hoc* (c'est le cas pour les boucles). C'est avec l'aide de ces registres que s'effectuent les opérations arithmétiques. Le plus utilisé pour ces dernières s'appelle l'**accumulateur**.

β - registres d'adressage

Ils contiennent l'adresse-mémoire des instructions ou des données. Cette adresse exige souvent deux ou trois registres et même le double dans certains travaux courants (3 registres pour l'adresse de lecture et 3 pour celle d'écriture par exemple dans les manipulations de chaînes). On expliquera à la page suivante (§ 2a) la raison de la complexité de cet *adressage.*

Les deux registres d'adressage les plus importants sont ceux donnant l'adresse de la prochaine instruction à exécuter. Souvent désignés par les symboles CS:IP (pour *code segment + instruction pointer*), ce sont eux que l'UAL consulte pour aller chercher en mémoire centrale le texte de l'instruction à exécuter et la soumettre au décodeur. Après son analyse, le décodeur indique de combien de pas-mémoire l'UC doit *incrémenter* le pointeur d'instruction IP (c.-à-d. lui ajouter) pour fabriquer l'adresse de l'instruction suivante.

γ - registre d'état

Il contient, sous forme d'*indicateurs (balises, flags)*, des informations précieuses sur le résultat des dernières opérations effectuées : indicateurs de *retenue*, de *parité*, de *débordement*, d'*erreur*, de *zéro*, de *signe*, ... modifiés *(armés)* par certaines opérations afin que les suivantes puissent exploiter leur résultat. On transmet de cette façon la retenue (*carry* en anglais) d'une addition partielle à la suivante, ainsi que le résultat d'une comparaison. Chacun de ces indicateurs **(1)** est formé d'un seul bit.

(1) Le registre d'état est parfois appelé – ou fait partie de ce qui est appelé – le PSW, *programme status word*, dans certaines machines.

2 - LA MEMOIRE CENTRALE

a - Mémoire vive

Physiquement, la *mémoire vive* est formée de plusieurs puces. Logiquement, elle constitue un vaste ensemble de cellules, chacune étant capable de conserver la valeur d'un bit. L'UAL écrit et lit en elles données et instructions. Les cellules ne sont pas toutes accessibles séparément. La mémoire s'articule souvent en *octets*, ensembles de 8 bits. Un octet possède une adresse : on dit qu'il est *accessible*. Cette adresse étant elle-même un nombre manipulé par l'UC, elle ne peut varier qu'entre des limites bien précises. Avec 8 bits de longueur, l'adresse ne peut s'échelonner que de 0 à 2^8-1 = 255, ce qui est vraiment trop peu. Aussi, l'adresse la plus simple comprend-elle toujours au moins deux octets, c.-à-d. 16 bits : elle est alors capable de "pointer" sur 65 536 mots-mémoire.

Ce nombre de mémoires, autrefois suffisant, n'est plus compatible avec l'ampleur des programmes actuels. C'est pourquoi l'adresse occupe maintenant 4 mots de 8 bits, ce qui lui permet d'atteindre théoriquement jusqu'à 2^{32} mots-mémoire, c.-à-d. près de 4,3 milliards d'octets !

Beaucoup de calculateurs personnels en service n'étant capables de manipuler *en bloc* que des mots de 16 bits, il faudra associer deux registres pour fournir les 32 bits nécessaires à l'adressage mémoire. Pour cela, on considère que cette mémoire est formée de **segments** juxtaposés (on aurait pu dire de *pages*) comprenant chacun 65 536 mots pointés par un registre, tandis que l'adresse du segment est donnée, elle, par le registre associé appelé *registre de segment*.

On a déjà énoncé que l'adresse de l'instruction à exécuter était donnée par le couple de registres CS:IP. Le symbole *deux-points* indique ici que les registres CS et IP ont été associés pour donner, le premier, le numéro du segment et le second, l'adresse dans le segment de l'instruction désirée (on dit aussi *décalage*, *déplacement*, en anglais, *offset*). Cependant – nouvelle difficulté –, l'adresse absolue ainsi composée n'est pas forcément égale à 65 536 × [contenu de CS] + [contenu de IP] , ce qui serait logique.

Dans les IBM-PC sous DOS par exemple, l'**adresse absolue** est donnée par **16×[CS] + [IP]** (les crochets signifient *contenu de*) ; ceci entraîne deux conséquences : d'une part, l'adresse absolue peut être écrite de multiples façons, puisqu'en pratique, tous les 16 octets (2^4), commence un nouveau segment ; d'autre part, cette adresse sera limitée à 16×65 536 octets, soit un peu plus d'un million ($2^{20} = 2^{16} \times 2^4$), ce qui est un peu trop juste (2).

Ce que nous venons de dire pour l'adressage des instructions s'étend à l'adresse de n'importe quel mot-mémoire : il faudra associer un registre de segment à un registre de décalage pour aller pointer sur n'importe quel octet de la mémoire. Les données ne figurant pas forcément dans le même segment que les instructions, on aura toujours au moins deux registres de segment : le registre CS *(code segment)*, déjà mentionné, pointe sur le segment où sont contenues les instructions, tandis que le registre DS *(data segment)* pointe sur celui des données. Il peut même y en avoir d'autres.

b - Mémoire morte

Il s'agit en général d'une mémoire de type **ROM**. Physiquement, elle diffère profondément de la mémoire vive, comme on l'a vu dans le chapitre précédent. Mais logiquement, elle constitue, au même titre que la mémoire vive, un **sous-ensemble de la mémoire centrale** et obéit à la même segmentation. Elle doit être de même rapidité. L'UAL s'adresse aussi bien à l'une qu'à l'autre pour lire, mais évidemment, seulement à la mémoire vive pour écrire.

La mémoire morte contient des programmes installés une fois pour toutes par le constructeur. Pour ne pas rendre impossible l'amélioration d'un calculateur, on ne place dans cette mémoire que des informations dont on est sûr qu'elles n'auront pas besoin d'être modifiées au cours de la vie de la machine. Le gros handicap de ce type de mémoire est son caractère définitif. On y trouve implantés généralement le programme de démarrage *(bootstrap)*, ceux de traitement de certaines erreurs, ceux de gestion de périphériques et, **parfois** (mais pas dans les IBM-PC, ni dans les Macintosh), le logiciel, appelé *interpréteur*, chargé de traduire en langage machine les commandes frappées au clavier par l'utilisateur.

(2) Cette limitation ne provient pas des UC actuelles, mais de la volonté chez les constructeurs de rester compatible avec les programmes écrits sous DOS pour les unités de type 8088. Les puces actuelles peuvent, par basculement d'un bit dans un registre, travailler soit en *mode réel* (comme sous DOS), soit en *mode protégé* (comme avec OS2, Unix ou Windows) avec alors un adressage de 2^{32} octets.

c - Antémémoire

Il s'agit là d'un module introduit pour accélérer les échanges dans les ordinateurs les plus performants. Appelée également *mémoire-cache*, elle ne fait pas partie physiquement de la mémoire centrale, dont elle serait plutôt le relais. De "petite" capacité et d'accès très rapide parce qu'implantée dans l'UC elle-même, elle contient la copie de la *page* (la zone) de la mémoire d'où ont été extraites les dernières informations. Les statisticiens ont en effet montré que, dans plus de 80% des cas, les lectures et écritures postérieures à une opération donnée concernent des adresses proches de celle de cette dernière. D'où l'idée de transférer en bloc la page intéressée à proximité immédiate de l'UC. L'accroissement de la taille de cette mémoire (256 Ko) a beaucoup contribué à augmenter la puissance des machines récentes.

3 - LES CANAUX D'ECHANGE

On compte trois canaux d'échange rapide, souvent appelés **bus**, consistant physiquement en un *circuit imprimé* **(3)** d'au moins 30 conducteurs parallèles. Ces canaux constituent la voie de communication entre tous les éléments rapides de la machine (UC, mémoire, portes d'entrée-sortie...). Ces éléments rapides peuvent occuper plusieurs cartes électroniques : elles sont toutes connectées sur le ou les bus.

Dans les plus modestes ordinateurs, ces canaux sont de simples voies de liaison. Dans les unités plus puissantes, ils comportent des circuits dotés d'une **autonomie** suffisante pour gérer directement le transfert de blocs de données entre deux modules. L'UAL, après avoir déclenché le transfert, peut effectuer d'autres travaux. L'exemple le plus connu est celui de l'**accès direct mémoire** ou DMA *(direct memory access).*

a - canal d'adresse

Le premier de ces bus est celui d'adresse : il comporte (au moins) 16 conducteurs, en principe autant que les registres d'adresse comprennent de bits. Comme son nom l'indique, il véhicule l'adresse de l'élément avec lequel l'UAL veut entrer en relation. Chacun de ces éléments est relié au bus par un *registre d'adressage* (formé de circuits ET), caractérisé par une adresse propre. Ce registre laisse pénétrer les informations du canal de données dans le module qu'il contrôle si – et seulement si – l'adresse présente sur le canal d'adresse est identique à la sienne. Il se comporte comme une serrure ne laissant ouvrir une porte qu'en présence de la bonne clé.

b - canal des données

Le canal des données est formé de 8, 16, 32 conducteurs ou même plus. C'est lui qui véhicule données et instructions – c.-à-d. tous les mots autres que des adresses – de l'UAL vers la mémoire ou vers les portes d'entrée-sortie, ou vice versa. Un mot entier (de 8, 16 ou 32 bits), acheminé par ces conducteurs en parallèle, est transmis en un temps appelé **cycle de transfert** comprenant quelques cycles d'horloge. Une machine est qualifiée de "8", "16" ou "32 bits" selon le nombre de conducteurs de son bus de données. Beaucoup de données étant formées de 16 ou 32 bits (cf. chapitre VI), un calculateur de 8 bits ne pourra les manipuler qu'en utilisant deux ou quatre cycles de transfert. La rapidité d'une machine est donc fortement dépendante de la largeur **(4)** de son bus de données (plus que de la fréquence de son horloge).

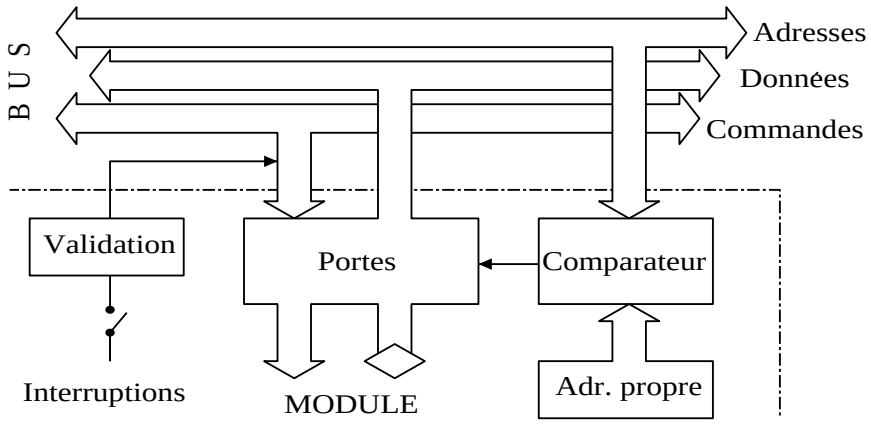


Fig. 4-3 : Système d'adressage d'un module d'entrée-sortie.

(3) La technique du circuit imprimé consiste à dessiner sur une carte isolante des conducteurs en cuivre par des moyens empruntés à l'imprimerie : insolation d'une résine protectrice à travers un cliché, puis dissolution des zones photosensibilisées et attaque chimique du cuivre découvert.

(4) Un bus *large* a pour contrepartie un grand nombre de broches sur l'UC, qui devient difficile à câbler (la puce 68020 de Motorola avec ses 32 bits possède 114 broches).

c - canal de commande

Un troisième canal, formé lui aussi de conducteurs parallèles, va véhiculer des **ordres** aux modules sélectionnés par l'UC. Ces ordres atteindront, comme il l'a été dit pour les données, uniquement le module *adressé*, c.-à-d. celui dont l'adresse est présente au même instant sur le canal d'adresse du bus. Parmi ces ordres, les plus fréquents sont ceux d'écriture et de lecture. Citons également [(5)] ceux d'entrée et de sortie – qui concernent les portes –, ainsi que ceux nécessités par les interruptions (dont on reparlera a la fin de cette page).

4 - LES PORTES D'ENTREE-SORTIE

En électronique, le mot *porte* (en anglais *gate*) désigne des circuits logiques élémentaires OU et ET. Mais en informatique, il désigne plutôt les circuits permettant d'accéder aux périphériques et composés de nombreuses portes binaires (les mots anglais lui correspondant sont alors *port* ou *driver*). En effet, l'unité centrale ne dialogue jamais directement avec un périphérique, celui-ci étant toujours beaucoup trop lent pour elle. L'UC n'accède qu'à des registres-mémoire, adressables comme la mémoire centrale et placés en général sur des cartes spéciales. Ce sont ces circuits qu'on désigne sous le nom de *ports* ou de *portes* d'entrée-sortie (en abrégé *portes* ES).

Pour réguler le flot des données entre le canal (rapide) et le périphérique (lent), on interpose souvent entre eux, outre les portes, des **mémoires-tampon** (*buffers* en anglais) qui régulariseront la transmission des informations (sauf si on les laisse déborder!).

Une remarque pour clore ce paragraphe sur les portes : dans beaucoup de machines (dont les IBM-PC), une même adresse dans le canal d'adresse peut aussi bien désigner une porte ES qu'un mot-mémoire. C'est le bus de commande qui lève l'ambiguïté : si l'UAL veut dialoguer avec la mémoire, elle excite la ligne *écriture* ou la ligne *lecture*; si elle demande une entrée-sortie sur un périphérique, elle excite la ligne *entrée* ou *sortie* du même bus.

5 - LE BUS LOCAL

La notion de bus est plus complexe que le suggère la description de la section 3. Au sein de l'UC, c'est un *bus interne* (cf. fig. 4-1) qui sert de lien entre ses constituants très rapides. Sur les IBM-PC, jusque vers 1992, le *bus externe* – i.e. hors de l'UC – reliait cette dernière aux mémoires et aux périphériques. Mais ce bus s'est avéré vite trop lent, surtout pour le graphique, qui exige de forts débits d'information. Il y a eu des tentatives pour remplacer ce bus par un bus externe plus rapide, mais les grands fabricants ne sont pas parvenus à s'entendre (cf. chap. XV, § 3, p. 131). D'autres fabricants ont décidé alors de résoudre le problème par le dédoublement du bus.

Avant le bus externe, on implante un bus intermédiaire, appelé *bus local*, bien plus large (32 bits) et plus rapide (33 Mhz) que le bus externe (16 bits, 8 Mhz). Il relie l'UC à la mémoire et aux périphériques les plus exigeants, comme la carte graphique, le disque dur, la carte sonore. On verra, chapitre XV, qu'un bus doit être normalisé. Aussi, à côté des bus locaux appelés *propriétaires* (propres à un fabricant d'ordinateur), sont apparus des bus normalisés comme ceux de type VESA ou VLB (VESA local bus) ou PCI. Le bus PCI (concepteur Intel) est plutôt destiné au Pentium et les bus VESA au 486. Des circuits de gestion donnent au bus local une grande autonomie ; c'est lui qui alimente le bus externe, soulageant d'autant le travail de l'UC. La présence d'un bus local améliore considérablement les prestations de la machine, surtout en ce qui concerne le graphisme.

6 - LES CIRCUITS DE GESTION DES INTERRUPTIONS

Certains appareils peuvent exiger que l'UC arrête le programme en cours et accorde toute son attention à une situation anormale (*exception* en anglais). On désigne ainsi un événement, certes prévisible et même prévu, mais dont on ne peut pas savoir à quel moment il se produira : on dit qu'il est aléatoire.

On va donner **deux exemples** de ce genre d'événements. L'un concernera le calcul : s'il apparaît une **division** avec un **zéro** en dénominateur, l'UAL ne sait que faire, les mathématiciens n'ayant pas encore donné de sens à ce genre d'opération. Bien sûr, tout bon programmeur doit prévoir cette éventualité et écrire les instructions permettant d'éviter alors la division.

(5) Une ligne de ce canal indique, sur les machines à adressage mixte, si l'on est en mode 16 ou en mode 32 bits.

Mais les concepteurs de l'UC ont prévu en quelque sorte une seconde ligne de défense pour pallier la négligence de certains programmeurs. Dans cette éventualité, une *interruption* est déclenchée : elle permet à l'UAL d'aller chercher à l'endroit voulu les consignes **(6)** adaptées à la situation (dans ce cas précis, elle affichera un message tel que *erreur fatale, division par zéro*, ou bien le numéro de l'erreur et arrêtera là l'exécution du programme). Cet exemple concernait ce qu'on peut appeler une *interruption interne*, puisqu'elle est provoquée par l'UC elle-même.

Autre exemple de situation accidentelle, mais cette fois-ci *externe* à l'UC : un événement fortuit sur un appareil piloté par l'ordinateur, par exemple un **manque de papier** dans l'imprimante. Celle-ci active alors une ligne donnée de son câble de liaison ; le signal parvient à la carte sur laquelle est câblée la porte d'accès à l'imprimante. Une interruption est alors déclenchée provoquant l'affichage d'un message à l'écran et interdisant tout nouveau transfert de données vers l'imprimante jusqu'au retour à la normale.

Les **situations accidentelles** prévues sont assez nombreuses (leur nombre caractérise l'aptitude de la machine à piloter de l'appareillage externe complexe). Un bon nombre de périphériques utilisent intensivement les interruptions : le clavier, la souris par exemple envoient à l'UC un signal d'interruption dès que l'opérateur agit sur eux. Ces interruptions peuvent même survenir **simultanément**. Pour en assurer la gestion correcte, les portes ES correspondantes excitent l'une des lignes du bus de commande réservées aux interruptions. Ces lignes parviennent à un **circuit de gestion**, qui examine leur validité, détermine celle à prendre en compte en priorité, met en attente les autres et transmet à l'UC en bon ordre les diverses demandes. Dans les IBM-AT, les deux circuits de gestion des 15 interruptions disponibles s'appellent des PIC *(programmable interrupt controller)* **(7)**.

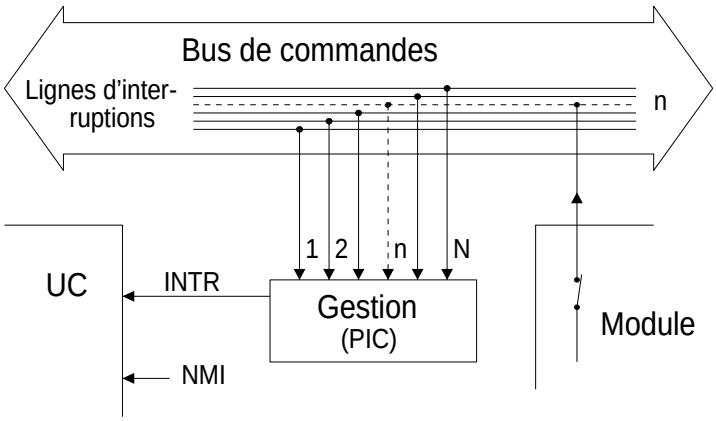


Fig. 4-4 : Système d'interruptions dans un IBM-AT.

L'appareil autorisé à déclencher une interruption est connecté au bus via une carte (porte, module), qui comporte des interrupteurs reliés à l'une des lignes d'interruptions du bus (ligne n*). La demande de l'appareil parvient ainsi au PIC, qui la compare à son masque, sorte de filtre programmable laissant ou non passer la requête* n*. Si elle est acceptée, le PIC excite la broche* INTR (interrupt request) *de l'UC, qui arrête le programme normal dès la fin de l'instruction en cours. L'UC demande ensuite au PIC le numéro n de l'interruption. Celui-ci la lui retourne via le bus adresse. L'UC se déroute alors à l'adresse absolue* 4(n+8) *de la mémoire centrale, où doit se trouver le début du* vecteur d'interruption *correspondant, adresse du programme chargé de résoudre le cas posé.*

La ligne INTR *est masquable. Cela veut dire que le programmeur peut interdire au PC de prendre en compte les demandes d'interruption lui parvenant. Par contre, il ne pourra pas (en principe) empêcher le déclenchement d'interruptions internes, hautement prioritaires, parvenant à l'UC par la broche* NMI (non masquable interruption).

(6) Dans les IBM-PC sous DOS, ces consignes sont contenues dans un programme appelé *vecteur d'interruption.*

(7) Dans les PC, les numéros d'interruption matérielle (ou IRQ) sont ainsi attribués : 0 à l'horloge de l'UC, 1 au clavier, 2 à la sortie du PIC n°2 ; puis en général : 3 à COM2, 4 à COM1 (cf. ch. XIV), 5 à LPT2 (2e imprimante), 6 au lecteur de disquette, 7 à l'imprimante, 12 à la souris type PS2, 14 au disque dur. Si l'on doit connecter une carte additionnelle à une interruption, ne jamais utiliser les 3 premières. L'emploi des IRQ 10, 11 et 15, non utilisées, est bien préférable, mais elles sont rarement disponibles sur les cartes.

EXERCICES

1 - A quoi sert l'unité centrale ? Donnez les autres noms sous lesquels elle est connue. Citez ses principaux modules.

2 - Qu'appelle-t-on un registre, les registres ? Quel est le rôle de ceux-ci ? Comment l'UAL compose-t-elle l'adresse d'un mot mémoire ?

3 - Pour quelle raison, dans un calculateur de 8 ou de 16 bits, l'*adressage* est-il complexe ? Quelles sont les limitations sur le volume de la mémoire ainsi introduites ? Quel gain dans ce domaine procure un calculateur de 32 bits ?

4 - Pouvez-vous expliquer comment se calculent les adresses dans les IBM-PC ? Conséquences ? Cette limitation provient-elle des circuits ? Sinon, de quoi ?

5 - Qu'est-ce que le bus ? Ses subdivisions ? Que faut-il sur les bus pour qu'une information parvienne à son destinataire ?

6 - A quoi sert le dispositif des *interruptions* ? Comment une information accidentelle extérieure parvient-elle à l'UAL et que se passe-t-il alors ?

7 - Qu'est-ce que le *registre d'état* ? En quoi est-il spécial ? Subdivisions et rôle ?

8 - Que signifie l'abréviation CS:IP ?

Chapitre V

LES PERIPHERIQUES

Les périphériques constituent des organes indispensables interposés entre homme et machine pour permettre entre eux une communication réciproque. On les appelle quelquefois *interfaces*, bien que ce mot ait de multiples significations. Ils sont reliés au cœur de l'ordinateur par les portes d'entrée-sortie et le bus externe ou bien le bus local. On va les classer en trois grands groupes, que nous étudierons à tour de rôle :

– les mémoires de masse,
– les périphériques d'entrée,
– les périphériques de sortie.

On notera l'importance des unités de lecture-écriture magnétiques qui entrent dans les trois catégories ci-dessus. Mais on parlera également de périphériques très populaires comme le clavier, les écrans, les imprimantes, la souris ... D'autres périphériques moins répandus seront aussi évoqués et nous terminerons par une section consacrée aux échanges sonores entre homme et machine. Ce type de communication, encore marginal en 1995, est en effet en évolution rapide.

1 - LES MEMOIRES DE MASSE

On appelle mémoires de masse des supports d'information de grande capacité. Leurs cellules élémentaires ne sont alors plus constituées de *circuits discrets* (transistors), mais de microscopiques domaines d'un matériau déposé sur un disque ou sur une bande et défilant sous une *tête de lecture-écriture*. L'avantage du procédé réside dans le faible coût de la cellule-mémoire élémentaire, son inconvénient dans le fait qu'on n'*accède* à une donnée que lorsqu'elle passe sous la tête. La capacité de ces supports s'exprime en kilo ou en mégaoctets (en abrégé ko, kø, Mo, Mø). En sus de leur **capacité**, ils sont caractérisés par leur **temps d'accès**, défini comme le temps moyen nécessaire pour aller lire ou écrire à une position arbitraire du support. Ce temps est bien supérieur au temps d'accès des mémoires centrales (qui est de quelques dixièmes ou de quelques centièmes de microseconde). Il se compte, lui, en dizaines de millisecondes pour les plus rapides, mais avoisine l'heure pour les plus lents.

Il y a 20 ans, il aurait fallu insister sur les **bandes** et **cartes perforées** ainsi que sur leur *codage*. Leur papier ou leur carton était "troué", rangée après rangée, par des poinçons et lu par des palpeurs mécaniques ou par des cellules photo-électriques. Dans chaque rangée, appelée *colonne*, on perforait et on lisait plusieurs trous en même temps (5 à 9 pour les rubans, cf. figure 5-1, 12 pour les cartes de type IBM). Ces documents étaient très encombrants, peu fiables, mais surtout les machines qui les manipulaient (perforatrices, trieuses, classeuses, lecteurs) possédaient de tels inconvénients (bruit, prix, encombrement, fragilité...) qu'ils n'ont pas survécu à la concurrence de la disquette.

a - Disquettes

Les disquettes *(disques souples, floppy disks)* constituent les plus populaires et les moins chères des mémoires de masse. Leur fonctionnement repose sur le magnétisme, tel qu'on l'a décrit dans le chapitre III. Une couche de poudre magnétique, dispersée dans un liant, a été déposée uniformément sur un disque *plastique*, qui tourne sous un rail fixe de lecture-écriture. Le long du rail, coulisse la *tête*, minuscule bobinage entourant un noyau de *ferrite*, capable de ne magnétiser qu'une surface microscopique du disque.

Il existe plusieurs sortes de disquettes, variant selon :

– le diamètre : 8" il y a 10 ans, 5"¼ et 3"½ actuellement,
– le nombre de faces sensibles : une ou deux,
– la finesse du grain, donc la *densité de bits* enregistrés **(1)**,

(1) Se souvenir qu'une *haute densité* est plus dense qu'une *double densité*! La densité *linéique* utile varie avec le rayon: la valeur nominale est celle de la piste centrale. Le formatage se faisant à densité *angulaire* constante, les pistes externes sont donc sous-employées.

– leur capacité, allant de 360 kø à 2,8 Mø .

La disquette tourne dans son étui; elle est lue ou magnétisée par la tête grâce à une fente ménagée dans l'étui (figure 5-1). On peut interdire l'écriture sur une disquette en basculant un ergot (sur celles de 3,5") ou en masquant une encoche (sur celles de 5"¼).

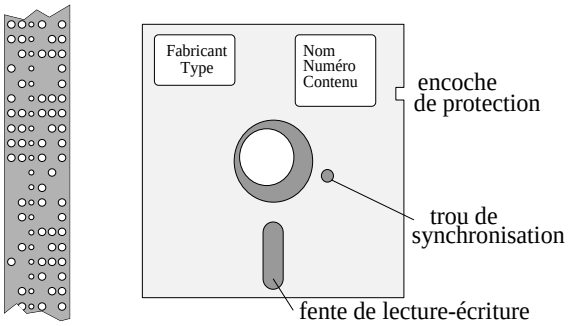


Fig. 5-1 : Ruban perforé et disquette de 5,25".

L'information ne peut s'inscrire que sur des **pistes** *(tracks)* circulaires concentriques, divisées en **secteurs**. Une disquette de 3"½ , de *haute densité* et double face, par exemple, comporte 2 fois 80 pistes de 18 secteurs de 512 octets chacun, soit plus de 1,47 Mø. On apprendra plus loin qu'on mémorise un texte à raison d'un octet par caractère : une disquette de ce type peut donc contenir en gros 250 pages d'un livre comme celui-ci (texte brut, hors édition, hors figures, cf. exercice n°1, chap. VI).

Connaissant les rayons extrêmes de la couronne utile sur la disquette (42 et 20 mm environ), on peut estimer à 0,2 mm la valeur de l'intervalle entre pistes et à environ 14 microns la longueur occupée par un octet sur la piste intérieure, la plus dense (on rappelle qu'il s'agit de 8 bits). La valeur élevée de l'intervalle entre pistes provient du manque de précision dans la translation de la tête sur son rail.

La disquette est insérée dans une *unité* de lecture-écriture appelée également *lecteur*. Celle-ci est mise en rotation à chaque demande d'utilisation. Elle s'arrête environ 2 secondes après la fin de l'opération. On gagne donc à grouper les appels à une même disquette. La tête de cette unité est en contact direct avec la disquette. L'usure de l'une et de l'autre est donc, à la longue, prévisible.

b - Le disque dur

D'une très grande capacité (30 Mø est un minimum et on dépasse parfois 900 Mø), le disque dur *(hard disk)* est formé de plusieurs plateaux métalliques montés sur un même axe.

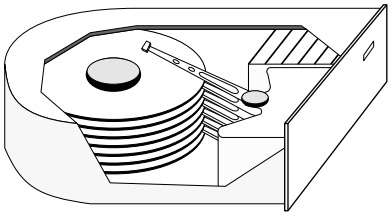


Fig. 5-2: Disque dur schématisé.

Chaque plateau est recouvert sur ses deux faces de poudre magnétique et agencé, comme la disquette, en pistes et en secteurs (fig. 5-2).

Le disque est en rotation **permanente** rapide (80 tours par seconde par exemple) dans un **boîtier scellé**, à l'abri des poussières et des polluants. Il comporte autant de têtes que de faces magnétisées ; ces têtes sont montées deux à deux sur un peigne pivotant dont les dents s'encastrent entre les plateaux. Le déplacement du peigne de piste à piste est plus lent que celui du disque sous la tête. Les temps d'accès moyens diffèrent donc selon le genre de déplacement effectué. Le long d'un même secteur, il varie par exemple de 10 à 80 millisecondes selon les modèles. On appelle **cylindres** les pistes de même diamètre contenues sur les différents plateaux. Le temps d'accès d'une piste à l'autre dans un même cylindre est très court, puisqu'il ne nécessite qu'une commutation électronique entre deux têtes différentes. C'est pourquoi l'enregistrement se fait cylindre par cylindre.

La distance tête-plateau doit être inférieure à la largeur occupée par un bit (fraction de micron). Pour y parvenir, la tête *vole* sur un microscopique coussin d'air à 0,3 micron des plateaux. Quand on connaît la dimension d'un grain de fumée de tabac (plus de 10 microns), on n'est pas surpris des mesures prises pour isoler le disque de l'ambiance. Des **précautions** similaires, quoiqu'un peu moins rigoureuses, devraient être observées par quiconque souhaite garder ses disquettes en bon état.

Malgré ce luxe, les disques durs peuvent s'endommager. Cet événement est même une mini-catastrophe que tout informaticien sérieux doit prévenir. Pour en amoindrir les effets, il existe une parade qui consiste à **copier** périodiquement le disque dur sur un autre support *(sauvegarde, back-up)*. En cas de mise hors-service du disque, on récupérera ainsi l'information qu'il contenait, sauf celle introduite depuis la dernière copie. On suggère qu'un disque dur ne devrait pas contenir le résultat de plus de 5 à 6 heures de travail non recopiées.

Selon l'importance du calculateur, on trouve dans la "**configuration**" plusieurs *unités* de disques durs et de disquettes. C'est le *système d'exploitation* (cf. chapitre VIII) qui en limite le nombre.

c - Bandes et cassettes magnétiques

Les **bandes magnétiques** *(tapes)* font preuve d'une **capacité** gigantesque (plus du gigaoctet), mais d'un piètre temps d'accès. La lourdeur et le prix de leurs *dérouleurs* restreignent aux gros ordinateurs l'emploi des bandes larges (plus de 2 cm, 250 bits/mm sur 9 à 17 pistes parallèles) montées sur de grandes bobines. Leur petite sœur, la **cassette** (petite bande en boîtier), qui était de règle sur les premiers calculateurs de bureau, a été détrônée par la disquette. Mais elle a pris sa revanche après avoir fait peau neuve : d'une taille et d'une qualité supérieures, elle peut contenir de 40 à plus de 100 Mø. Son temps d'accès se compte en dizaines de minutes, ce qui limite son emploi. Elle sert surtout à *sauvegarder* le disque dur. Cette sauvegarde peut exiger plus d'une demi-heure, mais elle est facile, car automatique. La même sauvegarde sur de nombreuses disquettes est beaucoup plus laborieuse. Certains lecteurs-enregistreurs de cassettes *(streamers)* comportent une tête oscillante, balayant la bande en travers du mouvement, comme dans les magnétoscopes. Tous les supports de cette section sont du type à *accès séquentiel pur* : pour accéder à une information, on doit lire tout ce qui la précède.

d - Formatage

Presque tous les supports d'information magnétiques nécessitent une préparation avant emploi : l'**initialisation** ou ***formatage***, faite dans l'unité elle-même sur la demande de l'utilisateur. Elle consiste à inscrire sur le support des marques magnétiques le divisant en pistes et en secteurs. Cette opération efface toutes les données déjà inscrites et ne doit donc être déclenchée qu'à bon escient (jamais sur le disque dur). Les mêmes disquettes ou cassettes (vierges) peuvent être employées sur différentes familles de calculateurs, mais, une fois *formatées* par un *système d'exploitation* **(2)**, elles ne sont, sauf exception, utilisables qu'avec un système de même type ou bien *compatible* avec lui **(3)**.

e - Autres supports

Mentionnons quelques autres types de mémoire de masse :

– le **tambour magnétique**, dont la très grande vitesse de rotation (jusqu'à plus de 2000 tours par seconde) lui conférait un temps d'accès d'une dizaine de microsecondes. Délaissé actuellement, il servait autrefois plutôt de mémoire vive que de mémoire de masse ;

– la **carte magnétique**, support bon marché utilisé par du personnel non spécialisé (le public) pour conserver une information très limitée (cartes bancaires, laissez-passer...). Elle a constitué la seule mémoire permanente des premières calculettes HP programmables ;

– le **disque optique**, dans lequel on a inscrit (par brûlure ou fusion avec un laser) des empreintes de la dimension du micron, détectables à la lumière, identiques à celles des **disques numériques** musicaux lus par diode laser (abusivement appelés *disques compacts*). La capacité de ces disques est gigantesque et frôle le gigaoctet : ils peuvent contenir autant d'information, images incluses, qu'un gros dictionnaire. Pour l'instant, il s'agit encore de mémoire morte (**CD-ROM**), dont le temps d'accès avoisine la seconde, mais une variante technique rend certains d'entre eux accessibles en écriture, tout au moins une fois (disques **WORM** : *write once, read many*). Bientôt apparaîtront des disques optiques complètement *réinscriptibles* : leur principe sera alors magnéto-optique (procédé évoqué à la fin du chap. III).

2 - LES PERIPHERIQUES D'ENTREE

C'est par eux que l'utilisateur d'un calculateur introduit auprès de l'unité centrale aussi bien l'information à traiter que les programmes de traitement. La plus grande partie des messages transite cependant par des tampons de mémoire vive avant d'accéder à l'UAL ; sinon cette dernière, qui travaille beaucoup plus vite que les systèmes d'entrée, serait considérablement ralentie.

(2) De plus, le *système* inscrit des informations spécifiques (type de système, capacité du support ...) dans les tout premiers octets du disque.

(3) Citons la possibilité de lire les disquettes DOS 3¼" sur les dernières versions du Macintosh munies du lecteur *Superdrive* grâce à l'utilitaire AFE *(Apple File Exchange)*. Attention, cet utilitaire ne convertit que les octets ou les chaînes, pas les nombres (cf. chap. VI et note 3, p. 44).

a - les lecteurs (magnétiques et optiques)

Les **lecteurs** de toutes les mémoires de masse décrites ci-dessus constituent les systèmes d'entrée les plus utilisés *(unités* de disque dur, disquettes, bandes, disque optique, cartes ...). Nous n'en reparlerons pas.

b - le clavier

C'est par lui que l'utilisateur introduit ses ordres et que le programmeur écrit ses programmes. Le clavier *(keyboard)* dérive de celui des machines à écrire, mais comporte plusieurs touches supplémentaires dont :

– le *pavé* numérique, bloc de touches servant à frapper les chiffres,
– les touches de déplacement du *curseur* (flèches),
– les touches de *fonctions*, auxquelles concepteur et programmeurs peuvent affecter des rôles spéciaux,
– les touches d'*altération*, qui, telles celle des majuscules *(shift)*, modifient la signification des autres touches et offrent ainsi des jeux de caractères supplémentaires. Trois ou quatre touches jouent ce rôle : la majuscule, déjà citée (même effet que sur les machines à écrire), les touches notées CTRL *(control)*, ALT ou bien celles notées ⌘ ou dans les Macintosh. Parfois, de nouveaux jeux sont fournis par combinaison de ces touches spéciales.

Le clavier est lu *(scruté)* en permanence, rangée par rangée, par un petit automate. Lorsqu'on appuie sur une touche, cet organe détecte les numéros de la colonne et de la ligne contenant la touche *(code géographique)*. Il génère alors un code clavier (un nombre) qui est mémorisé dans un tampon. Sur ordre du programme, l'UAL demandera au *système d'exploitation* de venir chercher ce code **(4)**. Le *système* le lui transmettra, mais après transcodage. Ce cheminement complexe permet à des claviers banalisés de s'adapter aux alphabets de différents pays **(5)**.

Certaines installations peuvent comporter plusieurs claviers et donc admettre plusieurs utilisateurs. Elles sont alors qualifiées de *multiposte*. Chaque poste comprend également un écran et l'ensemble clavier-écran s'appelle **console** ou *terminal*. Cette disposition n'a d'intérêt que si l'UC peut exécuter plusieurs *tâches* en même temps.

c - autres lecteurs

D'autres périphériques peuvent être branchés sur le calculateur pour lui apporter des informations. Beaucoup lui sont alors reliés par des systèmes d'entrée-sortie "à tout faire", les voies de *communication série ou parallèle* dont on reparlera (chap. XVI).

Tel est le cas de la **souris** *(mouse)*, petit boîtier muni d'une boule et d'un à trois boutons. On la pousse sur une table. En roulant, la boule produit des impulsions électriques déplaçant par à-coups un signe spécial sur l'écran, le *curseur souris*. L'opérateur amène le curseur à l'endroit désiré, puis il *clique* (il appuie sur l'un des boutons). A chaque impulsion, une interruption est déclenchée et l'UAL ajuste les coordonnées du curseur ou détecte la position des boutons pour orienter le travail en tenant compte du programme.

Tel est le cas également des **numériseurs** d'images *(digitizers)*. Les plus simples se déplacent à la main sur une tablette quadrillée recouverte d'un dessin ou d'un dessin ou d'un plan ; quand le curseur atteint le point désiré, on *clique* comme avec la souris.

Cet appareil permet d'introduire dans le calculateur les coordonnées des points principaux d'une image pour permettre au programme de l'étudier ou de la reconstruire.

Les **analyseurs** d'image *(scanners)* fonctionnent, quant à eux, comme des *bélinographes (fac-similé* ou *fax)* : ils balayent toute l'image point par point selon les deux coordonnées et transmettent pour chacun d'eux au calculateur son *niveau de gris* ou même éventuellement sa couleur.

(4) Le code-clavier est égal au code ASCII du caractère, si la touche frappée correspond à un caractère ASCII. Sinon, le code est formé de deux nombres successifs : le nombre zéro, suivi d'un numéro spécifique à la touche (touches curseur, touches de fonction, etc). Seuls les programmeurs ont à connaître cette particularité.

(5) Quel qu'il soit, le passage d'un clavier à un autre ne nécessite qu'une commutation logicielle au démarrage du calculateur vers une table traduisant en codes-clavier la position des touches *(code géographique)*. Le clavier anglo-saxon s'appelle QWERTY, du nom de ses premières lettres. Le clavier le plus utilisé en France est l'AZERTY (même remarque). De telles dispositions de touches sont des séquelles de théories sur la dactylographie, mais n'ont aucun intérêt pour la majorité des informaticiens, qui tireraient un bien meilleur parti d'une disposition alphabétique. Sur les IBM-PC, on peut commuter le clavier AZERTY en clavier QWERTY par Ctrl-Alt-F1 (et vice versa avec Ctrl-Alt-F2).

Cette liste d'organes d'entrée raccordables aux *portes de communication* ou au *bus externe* n'est pas exhaustive. Ces voies d'accès sont suffisamment souples pour accepter toutes les nouveautés. Dans la section 4, on parlera des accès sonores et dans le chap. XV, des accès par réseau ou par des appareils de mesures physiques ou industrielles. Par ailleurs, des périphériques peuvent être installés dans le coffret même de l'UC et communiquer avec elle toujours grâce au bus et aux portes ES. Tel est le cas de l'**horloge** qui permet de connaître date et heure avec une précision satisfaisante (attention : il ne s'agit pas de l'*horloge* de l'UC). La particularité de cette horloge est d'être alimentée *(entretenue)* par une pile, pour continuer à fonctionner même calculateur éteint **(6)**.

3 - LES PERIPHERIQUES DE SORTIE

Il règne en ce domaine la même diversité que pour les organes d'entrée. De nouveaux venus viennent petit à petit se joindre aux deux plus connus, l'écran et l'imprimante.

a - les enregistreurs magnétiques

Comme pour les périphériques d'entrée, il convient d'inscrire d'abord au rang des organes de sortie toutes les **unités** (tel est leur nom) manipulant les supports magnétiques des mémoires de masse : disques, disquettes, cassettes... (cf. §1). Seul le disque optique non enregistrable – CD-ROM – ne peut pas être compté parmi les systèmes de sortie.

b - l'écran

C'est le périphérique le plus utile tant pour le programmeur que pour l'utilisateur d'un logiciel. Il permet de lire du texte ou de voir des dessins (*images* ou *graphiques*). On distingue actuellement trois types d'écran : *cathodique, à plasma, à cristaux liquides*. Mais il est possible que d'autres types apparaissent dans un proche avenir.

Nous détaillerons surtout l'écran cathodique. Les autres types permettent de fabriquer des écrans plats et légers, indispensables aux *calculateurs portatifs*. Ils sont peu lumineux pour l'instant et les modèles dotés de couleurs ne sont pas très satisfaisants. Objets de recherches intensives, ces écrans pourraient voir leurs prestations s'améliorer, à moins qu'un candidat inattendu vienne brouiller les cartes. Il pourrait en être ainsi de l'écran à *effet de champ* ou à *micropointes*, annoncé récemment comme très prometteur **(7)**.

α - Types d'écran

L'écran le plus utilisé en informatique est celui formé par la face avant d'un **tube cathodique** voisin de celui des récepteurs de télévision. Un *canon* émet un faisceau d'électrons très fin qui, après accélération sous plusieurs kilovolts **(8)**, bombarde un point de l'écran qui s'illumine par fluorescence (c.-à-d. émet une lumière propre à la substance déposée, le *luminophore)*. On obtient un écran-couleur en le compliquant (fig. 5-4) : on ajoute une grille à trous très fins appelée *masque de sélection des couleurs.* On dispose côte à côte trois canons, dont les faisceaux sont focalisés sur un même trou du masque : ce trou définit le point lumineux élémentaire *(pixel)*. Sur l'écran, on dépose trois types de luminophores, un par couleur fondamentale, sous forme de *triplets* minuscules, au point d'impact des faisceaux de chacun des canons. En modulant en plus ou en moins l'intensité de chaque canon, on donne au pixel la couleur désirée par

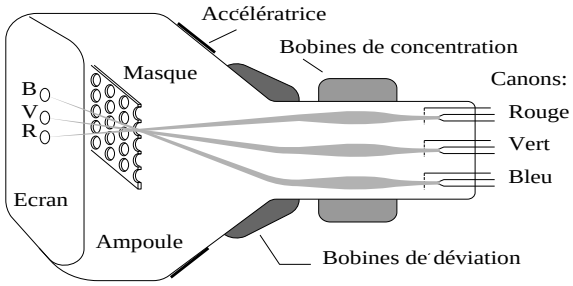


Fig. 5-4 : Ecran cathodique couleur.

(Les éléments ne sont pas à l'échelle : le masque est en fait tout près de l'écran, les impacts des faisceaux RVB pour un même pixel ne sont séparés que de quelques fractions de millimètre)

(6) Dans certains calculateurs, cette pile *(battery)* entretient également une petite mémoire RAM qui conserve, outre la date et l'heure, quelques informations sur la configuration de l'installation (type et nombre d'unités de disquettes, de disques durs ...). Cette mémoire est appelée CMOS, car elle est constituée de transistors de ce type (donc à effet de champ). Faible consommatrice d'électricité, elle est alimentée pendant plusieurs années par une pile au lithium.

(7) Ecran voisin des écrans cathodiques, avec des pixels trichromes excités par des électrons émis par des pointes microniques en silicium (plus de mille pointes par pixel pour minimiser l'irrégularité de l'effet de pointe individuel). Ces écrans plats, meilleurs et moins chers que ceux à cristaux liquides devraient être bientôt produits par l'industrie.

(8) Comme avec les écrans de télévision, ce bombardement produit des rayons X dangereux à la longue : ils doivent être atténués par un verre filtrant monté ou non en usine.

mélange des trois composantes fondamentales, bien que leur superposition ne soit qu'approchée (l'imperfection est visible à la loupe, placée près de l'écran).

L'écran **à plasma** est formé de trois plaques de verre superposées (fig. 5-5). Dans la plus interne, ont été percés de minuscules canaux transversaux contenant un gaz, du néon par exemple (donnant des écrans couleur **orange**). Sur les deux autres, soudées par la tranche à la première, on a imprimé un réseau de conducteurs, en ligne pour l'une, en colonne pour l'autre.

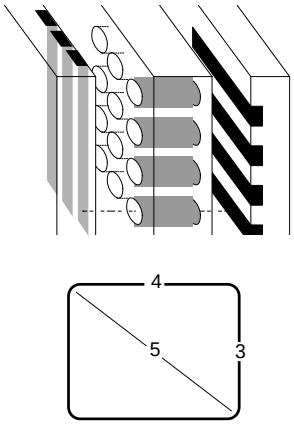


Fig. 5-5 : Ecran à plasma schématique et rapports des dimensions d'un écran.

Chaque canal débouche d'un côté sur une ligne et une seule, de l'autre sur une colonne et une seule. Une tension électrique adéquate sur une ligne et sur une colonne sélectionne un seul canal et y provoque une micro-décharge. On obtient ainsi le point lumineux élémentaire ou *pixel*, dont la taille et le nombre (au moins 650×350) sont les caractéristiques principales d'un écran. Le fonctionnement de l'écran à plasma est rapide, mais il consomme trop d'électricité pour qu'on puisse s'en servir sur batterie. Les calculateurs à écran plasma s'alimentent plutôt sur le *secteur* 220 volts.

L'écran à **cristaux liquides** utilise deux propriétés que possèdent certaines molécules *nématiques* : d'une part leur structure en hélice est sensible aux champs électriques et peut être détruite *réversiblement* par ceux-ci [(9)]; d'autre part, au repos, elles font tourner la polarisation de la lumière, de sorte que, placées entre deux polariseurs croisés (dispositif normalement opaque), elles rétablissent la transparence locale du système. L'application d'un champ électrique en un point donné, détruisant temporairement la structure nématique, rend donc ce point opaque (le milieu nématique joue un peu le même rôle que le milieu magnétique des disques optiques réinscriptibles, cf. fin du chapitre III, page 22).

Contrairement à l'écran plasma, l'écran à cristaux liquides n'émet pas de lumière : il utilise celle de l'ambiance ou celle d'un éclairage annexe. Ceci lui vaut d'être économique en électricité, d'où son emploi intensif dans les calculateurs portatifs autonomes. On lui reproche cependant son manque de luminosité, surtout en dehors d'un cône de visibilité assez étroit (35°), ainsi que sa tendance au traînage, effet dû à la lenteur des processus physiques impliqués.

(9) Pour les physiciens, il s'agit là d'un cas particulier du passage de l'ordre au chaos.

β - Résolution

Quel qu'en soit le principe, un écran couleur est cher. Il est d'autant plus difficile à construire qu'il comporte un plus grand nombre de *pixels*. Avec les écrans cathodiques, ce nombre dépend du pas du masque *(pitch)* et de la taille de l'écran. La norme VGA par exemple impose 640×480 pixels. Ce nombre sera atteint avec un tube cathodique de 12 pouces en diagonale (soit 23×20 cm environ) et un masque au pas de 0,35 mm (le pas des tubes de télévision actuels est de 0,5 mm). Pour obtenir un écran plus riche, tels ceux dits *haute résolution* ou *super-VGA*, il faut diminuer le pas. Un écran de 14" (largeur 26,5 cm) au pas de 0,28 mm ne peut comporter que 952 pixels ($14 \times 0{,}75 \times 25{,}4 / 0{,}28$). Il faut au moins un écran de 15" avec ce pas pour afficher 1020 points. Les "publicités" sont beaucoup plus optimistes!

γ - Constitution logique

Généralement, on peut utiliser l'écran selon deux modes différents. Le **mode graphique** est celui dans lequel le programmeur peut accéder directement à la totalité des pixels (ce mode est privilégié avec les UC de la famille Motorola 6800 et donc dans les calculateurs Macintosh). Le **mode caractère** ou **texte**, quant à lui, repose sur un matriçage (purement logiciel) de l'écran en n lignes de m colonnes (par exemple n = 20 à 25 et m = 80). Dans ce cas, on n'accède qu'aux nœuds de la matrice, mais on y place alors un caractère complet. Tout caractère est lui-même formé de pixels contenus dans une grille de 7×9 ou de 8×10 points. Toute la finesse de l'écran est donc correctement utilisée, mais on n'a accès qu'à des caractères prédéfinis par le concepteur du système. Le mode caractère est celui privilégié dans les UC de la famille Intel 8080 (appelée aussi 80*x*86).

Dans le cas de l'écran cathodique, ce sont des bobinages alimentés par des amplificateurs qui déplacent le faisceau électronique à grande vitesse : chaque point n'est éclairé que pendant moins d'un dixième de microseconde ; c'est la durée de la fluorescence (comme dans la télévision) et les propriétés de l'œil (comme au cinéma) qui convertissent ce balayage rapide en une image globale, fixe ou évolutive. L'écran, le tube cathodique, les amplis et alimentations associées sont réunis en un même boîtier souvent appelé **moniteur**.

Dans toutes les versions d'écran, les amplificateurs ou les circuits générant les balayages, reçoivent leurs signaux de circuits spécifiques à chaque mode de fonctionnement et implantés sur une carte dite *carte vidéo*. Cette carte comprend entre autres le générateur de caractères et une mémoire-tampon, appelée *mémoire vidéo* (sous-ensemble de la mémoire centrale) : la taille de cette mémoire dépend du nombre

de pixels *adressables* et de leurs attributs. C'est dans cette mémoire que les circuits de la carte vont chercher l'information indispensable (type, couleur, intensité) au moment de représenter pixel ou caractère.

La mémoire vidéo peut même conserver plusieurs images d'écran en mode texte. A raison de 2 octets par caractère – un pour son numéro ASCII, l'autre pour sa couleur –, un écran de 25 lignes à 80 caractères exige 4 kø. En mode graphique, il faut beaucoup plus de mémoire pour conserver une image (2, 4 bits ou plus par pixel, selon la richesse de la *palette*). Un dessin comportant beaucoup de points exige non seulement un écran de résolution suffisante, mais aussi une mémoire vidéo de taille correcte. On peut parfois utiliser différents *modes graphiques*, qui privilégient tantôt le nombre de pixels, tantôt la richesse de leurs coloris. Avec une carte VGA par exemple, on distingue une quinzaine de modes possibles, dont certains ne font qu'assurer la compatibilité avec les cartes plus anciennes (EGA, CGA). Une image VGA de 640×480 pixels avec 16 couleurs exige $640 \times 480 \times \log_2 16$ bits, soit 153,6 kø, car avec 4 bits, on peut obtenir 16 teintes par combinaison des trois couleurs fondamentales. Une image superVGA de 1024×728 pixels en 32768 couleurs exige près de 1,5 Mø. Alors que la mémoire de la carte VGA, située dans l'*espace adressable* du DOS, est facilement programmable, celle de la carte SVGA ne peut être programmée que par un adressage indirect (c.-à-d. par commutation de pages).

c - les imprimantes

Indispensables pour communiquer à autrui le résultat d'un travail, les imprimantes *(printers)* relèvent elles aussi de différents types. Signalons tout d'abord la machine à écrire électrique classique, à vrai dire peu utilisée par l'informatique, son jeu de caractères étant par trop réduit. Les machines à *boule* ou à *marguerite*, plus rapides, ont été un peu plus employées, mais l'accession à leur vaste jeu de caractères exige le changement de la tête d'impression. Cette intervention manuelle est peu compatible avec l'informatique.

Dans les centres de calcul, on a beaucoup employé des **imprimantes lourdes**, de vitesse élevée (jusqu'à 10 lignes par seconde), dont les caractères, portés par des chaînes, des tambours ou des barres, défilaient en permanence devant le papier. Des marteaux venaient frapper "au vol" les caractères voulus quand ils passaient à l'emplacement désiré. Le jeu de caractères de ces machines était très réduit et ne comportait souvent que les majuscules et les chiffres.

Les imprimantes précédentes comportaient des caractères "rigides", moulés dans la matière, comme ceux des machines à écrire. Les imprimantes maintenant les plus nombreuses sont de type **matriciel**. Leurs caractères sont formés de points, comme à l'écran. On a alors accès à un jeu étendu de signes et même à différentes *polices* de caractères. Plus les points seront nombreux et denses, meilleur sera l'aspect du texte. Ces machines peuvent même frapper deux fois de suite un caractère, avec un léger décalage, de façon à obtenir soit une meilleure qualité, soit le *graissage* (caractère gras). Beaucoup fonctionnent selon deux modes : le mode *épreuve*, rapide, mais grossier (les points, peu denses, sont encore visibles), le mode *courrier* ou LQ (*letter quality*), plus lent, mais produisant des caractères plus agréables à lire.

Les **imprimantes à aiguilles** sont de ce type. Leur tête est formée de minuscules aiguilles horizontales placées l'une au-dessus de l'autre dans un même plan vertical (fig. 5-6) et frappées par des marteaux. Le nombre d'aiguilles (7 à 24) conditionne la qualité et la rapidité du travail. Certaines machines dépassent la cadence de 300 caractères par seconde (en mode épreuve).

Dans l'imprimante à **jet d'encre**, la tête comporte de minuscules trous (buses) par où sont projetées de fines gouttelettes d'encre jusque sur le papier. Cette machine est particulièrement silencieuse, tout comme l'est l'imprimante **thermique**, dans laquelle de très fines aiguilles chauffées viennent *marquer* le papier.

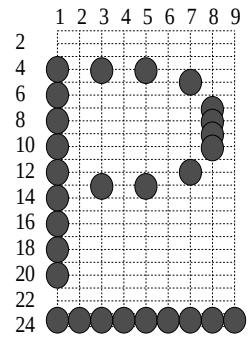


Fig. 5-6 : Lettre P *soulignée obtenue avec une imprimante matricielle à 24 aiguilles.*

Cette dernière est pénalisée par le papier *thermosensible* qui lui est nécessaire : il est cher et peu stable ; le texte risque de s'évanouir en quelques semaines s'il est exposé à la lumière et à la chaleur.

Parmi les imprimantes matricielles, celles **à laser** tiennent le *haut du pavé*. Le faisceau lumineux d'une diode laser **(10)**, *défléchi* par des miroirs rotatifs, dessine, point après point, ligne après ligne, les caractères d'une page sur un tambour en rotation recouvert d'un matériau photosensible. Ce tambour, ainsi que le matériel nécessaire à la suite des opérations (encrage du tambour, transfert de l'encre sur le papier, fixation au four par fusion de l'encre), est identique à celui des photocopieuses. La remarquable qualité de leur prestation provient de la densité des points qu'elles assurent : 300 par pouce (300 *dpi, dots per inch*) et même 600.

(10) Toutes ces imprimantes reposent sur l'électro-optique, mais ne comportent pas forcément une diode-laser. Leur appellation n'est donc pas toujours correcte.

La densité de 300 points par pouce est supérieure au pouvoir séparateur de l'œil à la distance de 20 cm. Aussi ne voit-on qu'un trait continu, sauf parfois dans les diagonales **(11)**. Il faut remarquer qu'on est encore loin d'atteindre la résolution des *photocomposeuses* des imprimeurs professionnels, lesquelles dépassent 2000 points par pouce, mais dont le fonctionnement est en partie analogue.

Les imprimantes de types ou de constructeurs différents font preuve de quelque **compatibilité** entre elles. Cette compatibilité repose rarement sur des normes, mais sur la prédominance d'un constructeur, avec lequel les autres ont dû s'aligner **(12)**. Il en est ainsi pour leur câble de connexion, qui, dans la majorité des cas, sera de type **Centronix**, ou moins souvent, de type RS232 ou GPIB (on reparlera de ces liaisons dans le chap. XV consacré à la télématique).

D'autre part, les imprimantes reconnaissent presque toutes les codes ASCII et, au reçu d'un tel code, impriment le caractère correspondant. Elles sont capables de beaucoup plus, par exemple de souligner, d'imprimer en **gras**, en *italique*, d'employer différentes polices, de tracer des lignes verticales, horizontales, de dessiner ..., autant de possibilités mal satisfaites par les caractères de commandes (numéro inférieur à 32) de la table ASCII. On utilise alors des séquences spéciales, dites **séquences d'échappement** (*escape*), formées d'au moins 3 codes normaux débutant par le caractère ASCII numéro 27 (cf. chap. XI, §6, page 97). L'arrivée de ce caractère oblige l'imprimante à appliquer aux signes qui suivent un traitement spécial (ce dernier paragraphe ne concerne pas les imprimantes de type PostScript, qui mettent en œuvre un procédé beaucoup plus complexe).

d - autres périphériques de sortie

On ne peut être exhaustif sur le sujet, tant ils sont nombreux. Citons la **table traçante** *(plotter)* dans laquelle des plumes de différentes couleurs (ou de différentes largeurs), sélectionnées par programmation, dessinent droites ou courbes sur le papier. Les mouvements du papier et de la plume sont assurés par des **moteurs pas-à-pas** ; le pas de déplacement atteint 50 à 25 microns. Il en existe pour tous les formats de papier, du A4 au A0. Ces appareils tracent schémas, graphiques ou plans... Il existe également des traceurs électrostatiques, sans organes mécaniques.

Quelques mots sur les **projecteurs** grand écran, capables de projeter à plusieurs mètres sur un écran de cinéma l'image agrandie de l'écran du calculateur. Le système dit *vidéo* comporte trois tubes cathodiques monochromes de forte luminosité (un tube couleur, à cause de la fragilité de son masque, ne supporterait pas la forte puissance nécessaire). Leurs images, colorées par filtres RVB, sont rendues convergentes par trois optiques différentes. Ce procédé, cher et délicat, fournit sur l'écran mural une image remarquable. Des appareils plus modestes utilisent des cristaux liquides dans une tablette translucide que l'on pose sur un *épidiascope* (projecteur de *transparents*). Le résultat est voisin, mais luminosité et rendu des couleurs n'atteignent pas ceux du premier procédé **(13)**.

4 - MESSAGES SONORES

Il reste à mentionner la possibilité d'échanges sonores entre homme et machine. La majorité des calculateurs comporte un petit haut-parleur destiné surtout à alerter l'usager lorsqu'il a exécuté une manœuvre interdite *(bip)*. Ce haut-parleur reçoit des signaux périodiques générés par des circuits programmables : dans les IBM-PC, ces circuits s'appellent PIT *(programmable interval timer)* et PPI *(programmable peripheral interface)*. Ce dernier n'est qu'un interrupteur reliant la sortie du PIT au haut-parleur. Quant au PIT, il est formé de 3 cascades de bascules (3 *canaux*) fonctionnant en diviseurs pour la fréquence horloge de la machine. Les deux premiers canaux produisent des signaux horaires de service (l'un à la fréquence de 18,2 c/s, l'autre assurant le rafraîchissement des DRAM). La valeur du facteur de division du troisième canal est programmable par l'utilisateur. On peut ainsi jouer sur la fréquence des signaux émis de façon à reproduire les notes de la gamme (le timbre toutefois n'est pas très riche). Pour fixer leur durée, on commandera le PPI à des intervalles fixés soit par une fonction de type retard, présente dans la plupart des logiciels de programmation, soit grâce à l'interruption prévue pour les mesures temporelles **(14)**.

(11) Une technique (RET ou similaire) permet une modulation de la taille du point telle que l'effet d'escalier dans les diagonales disparaît.
(12) Les constructeurs désirant entrer en compétition annoncent que leur machine en *émule* (se comporte comme) une autre plus connue.

(13) On pourrait voir apparaître des projecteurs à faisceaux laser, comme dans les spectacles (textes encore très courts).
(14) En Basic, l'instruction **SOUND** commande à la fois hauteur et durée des notes. L'instruction **PLAY** est encore plus puissante : elle permet de faire jouer toute une suite de notes (cf. p. 83).

Un ordinateur est tout à fait apte à piloter un émetteur de sons artificiels plus musicaux, comme un *orgue électronique*, ou un **synthétiseur** de musique. De même, on peut le faire suivre d'un **synthétiseur de parole**, comme celui de l'*horloge parlante* ou ceux de certaines voitures haut de gamme. Ces appareils génèrent des sons purement artificiels, ne provenant pas du tout d'enregistrements de la voix humaine, mais formés d'une suite de *tops* sonores, reproduisant les ondes pseudo-périodiques porteuses de la parole.

On verra dans le chapitre XVI (§6) l'importance prise récemment par le son et le *multimédia*. Au plan matériel, ces possibilités se traduisent par l'adjonction d'une carte sonore, éventuellement suivie d'amplificateurs musicaux, puis de haut-parleurs. Elle assurera les fonctions de synthèse sonore citées dans l'alinéa précédent.

Nous n'avons pas mentionné les **échanges sonores** dans la section consacrée aux périphériques d'**entrée**. Et pourtant il existe bel et bien des périphériques de ce type, mais ils n'ont pas beaucoup dépassé le stade expérimental. Si certains sont déjà opérationnels, c'est pour tenter de résoudre des problèmes autrement insolubles, par exemple pour permettre à un handicapé manuel d'utiliser un ordinateur. L'intérêt d'un tel organe est cependant très général et les périphériques d'entrée sonore remplaceront plus d'un clavier lorsque le procédé sera sans défaillance et acceptera un jeu suffisamment vaste de commandes. Pour en arriver là, il faudra maîtriser l'ensemble des problèmes posés par ce que l'on appelle la *reconnaissance vocale* (analyse harmonique de la voix humaine selon la théorie de Fourier).

Fin 1994, IBM a annoncé la sortie d'un périphérique qui permet de dicter du texte à un ordinateur. Il semblerait que le nombre d'erreurs soit encore significatif et il faut dicter en articulant syllabe par syllabe ; mais le progrès est notable et le gain de temps en saisie très important.

EXERCICES

1 - Les mémoires peuvent être représentées selon une certaine hiérarchie. Reconstituez cette hiérarchie, en les cataloguant en fonction de leur volume. On notera que leur temps d'accès varie en général dans l'ordre inverse du volume. Dans ce catalogue, repérez celles qui reposent sur le magnétisme, celles qui sont *volatiles* (perdent leur contenu à la coupure de l'alimentation) et celles dont on a dit que leur développement modifiera sans doute sous peu le marché des mémoires de masse.

2 - Comparez disquettes et disque dur.

3 - Qu'appelle-t-on *unité* de disque ?

4 - Citez des noms de périphériques d'entrée et résumez leurs caractéristiques.

5 - Mêmes questions pour ceux de sortie.

6 - Qu'avez-vous retenu sur le sujet de la carte vidéo ?

INTERLUDE

Nous avons rapidement étudié les circuits et les appareils composant une installation de calcul (ou *ordinateur*). Cependant, la puissance de cette *installation* et son adéquation aux problèmes qu'elle est censée résoudre dépendra tout autant des programmes associés que du matériel assemblé.

La deuxième partie de ce manuel sera consacrée à la façon de programmer ces appareils et à celle d'utiliser les programmes mis au point par d'autres. Cet aspect de l'ordinateur se nomme le **logiciel**.

Les Anglo-Saxons ont appelé la partie matérielle le **hardware**, mot qui signifie *quincaillerie*, et ont baptisé le logiciel **software** (et même *soft*), néologisme créé pour opposer les deux concepts grâce au jeu de mots entre *hard*, dur, et *soft*, mou. Doit-on en conclure que la programmation ou l'exploitation des logiciels se fait "en douceur" ou bien exige plus de doigté que la conception du matériel ? Il y a un peu de ça : les erreurs logicielles sont plus facilement réparables, on peut procéder par étapes et par essais. De plus, il est presque impossible de détruire l'appareillage par une fausse manœuvre logicielle (attention : cela ne veut pas dire ne pas perdre de l'information!).

On pourrait ajouter que le développement du matériel de calcul et même de ses périphériques ne peut guère être l'apanage que de sociétés puissantes, de *noyaux durs* réunissant des moyens financiers et humains importants. Par contre, dans le logiciel, de petites sociétés et même des francs-tireurs peuvent donner libre cours à leur créativité et obtenir de beaux résultats.

Dans la description du logiciel, on procédera par étapes, partant des niveaux les plus bas (c'est-à-dire les plus proches de la machine) pour accéder aux niveaux les plus élevés, les plus universels, les moins dépendants d'une machine ou d'un type de machine donné.

Dans les chapitres suivants, on imprimera en caractères **helvetica-narrow** les mots et textes constituant des instructions valides, telles qu'on peut les écrire dans un programme ou bien au clavier. Dans ces mêmes instructions par contre, un mot en *italique* indiquera une variable, c'est-à-dire une entité générale, symbolisant celles qu'on est autorisé à écrire à la même place.

Chapitre VI

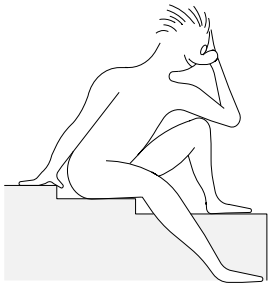
LES OBJETS DE L'INFORMATIQUE

Il ne faudra pas donner dans ce chapitre au mot *objet* le sens de *chose*, de concret, mais celui que lui donnent la philosophie, la logique ou la grammaire : entité à laquelle *s'applique* l'action ou la pensée, par opposition à *sujet*. En informatique, les objets seront les entités manipulées, donc tous les types de données et d'instructions.

1 - LE BIT

Le bit est la plus petite information manipulable, l'unité élémentaire de la numération, le plus petit nombre entier. Elle ne peut avoir pour valeur que *vrai* ou *faux*, *oui* ou *non*, 1 ou 0. Son nom est la contraction de *binary digit*. Le terme français **monade**, de même signification, est trop peu connu pour que nous pensions devoir le substituer à *bit*.

Autrefois (avant le règne de la télévision), on jouait parfois au jeu intellectuel (!) appelé *je pense*. Le meneur pensait à une personne, à un animal ou à une chose. Les joueurs tentaient de deviner l'*objet* choisi en lui posant des questions, auxquelles il ne il ne devait répondre que par oui ou par non. On avait compris que le meilleur moyen de faire durer le jeu était de ne céder à chaque question que le minimum d'information, un seul bit à la fois : *oui* ou *non*. Entre parenthèses, ce jeu constituait un excellent exercice de logique.



Je distille l'information bit à bit.

2 - L'OCTET

Le bit contenant vraiment trop peu d'information, on utilise beaucoup ses multiples, dont le plus important est l'**octet** (en anglais *byte)*, ensemble de 8 bits. La mémoire centrale est d'ailleurs organisée en octets, seuls *objets adressables*. L'UAL, quant à elle, manipule de 1 à 4 octets à la fois (et même 8 dans les gros calculateurs). On peut définir un sous-multiple de l'octet : le **quartet**, formé de 4 bits (en anglais *nibble)*. Hélas peu employé, il est cependant très commode, car il correspond exactement au contenu représenté par un chiffre hexadécimal. Comme on l'a expliqué à la fin du second chapitre, un octet peut recevoir, outre le zéro, 255 valeurs, c.-à-d. FF en base 16, un chiffre par quartet.

On se gardera bien de considérer l'*octet* comme le nombre de base de la numération *octale* : ce nombre est le 8 et se trouve représenté par 3 bits, groupement qui n'a pas reçu de nom, cette base étant maintenant peu employée. Par contre, on peut dire que l'**objet de base de la numération hexadécimale est le quartet** (4 bits, 16 valeurs).

3 - LE CARACTERE

L'écriture du français utilise une centaine de signes (symboles, **caractères**, *characters*) : les 52 lettres de l'alphabet latin, une quinzaine de *diacritiques* (lettres accentuées) et autant de signes de ponctuation. Quand on a voulu coder l'information, on a fait correspondre **un nombre** à chaque caractère. Le problème s'est d'abord posé pour le télégraphe électrique, après constat des difficultés que soulevait l'alphabet Morse (parce qu'en réalité c'était un code ternaire). On a d'abord codé lettres et chiffres **(1)** avec 5 perforations sur un ruban de papier (fig. 5-1) : on disait à l'époque 5 *moments*, on dirait maintenant 5 bits. Avec seulement $2^5 = 32$ symboles différents, ce code était trop pauvre : un même caractère devait représenter tantôt une lettre, tantôt un chiffre, selon la valeur d'un symbole préalable de sélection. Six bits étant encore insuffisants et la valeur 7 mal à son aise

(1) Ce code est dû à Emile Baudot, dont on a honoré la mémoire en donnant le nom de *baud* à l'unité de débit informatique sur une ligne.

dans un système binaire, on décida de consacrer un octet entier au codage d'un caractère. Actuellement, les mots **octet et caractère sont équivalents** dans presque tous les systèmes informatiques.

Dans un octet, on peut *loger* 2^{8} = 256 valeurs ou 256 symboles, si on attribue un numéro à chaque symbole. Cette correspondance a fait l'objet de normes, parfois de la part d'organismes internationaux : on a connu les codes CCITT (celui du télégraphe), EBCDIC (cher à IBM).... Actuellement, le code **ASCII** *(American standard code for information interchange)* est universellement employé. Il fixe la valeur de 128 symboles : les 32 premiers sont des **codes** de télécommunication ou de mise en page (n°13*: retour-chariot*, n°10: *à la ligne*, n°7: *attention*, n°6: *bien reçu*, n°8: *retour arrière*...) conçus spécialement pour la transmission par télex ; puis on rencontre les **chiffres** (codes 48 à 57 pour les chiffres de 0 à 9 compris), les **majuscules** (codes 65..90 pour A..Z) et les **minuscules** non accentuées (97..122 pour a..z); entre ces groupes, se placent les **signes de ponctuation**.

Dans cette table, les 128 "places" restantes n'ont jusqu'ici fait l'objet d'aucune norme. Mais il existe une table de correspondance très utilisée appelée **extension IBM au code ASCII**. Elle permet de représenter à peu près toutes les lettres accentuées issues de l'alphabet latin, quelques lettres grecques, des symboles commerciaux, mais aussi des caractères faits de segments de droites permettant de dessiner des cadres rectangulaires (à trait simple ou double) : ces derniers symboles, très utilisés, s'appellent "**caractères semi-graphiques** IBM". La figure 6-2 ci-après montre les cadres réalisables avec de tels symboles et donne leur numéro (pour les introduire au clavier, il suffit en général de maintenir la touche ALT enfoncée pendant qu'on frappe leur numéro de trois chiffres). La table de l'**annexe 1** (en fin de volume) présente le code ASCII dans son intégralité, plus ses extensions IBM et Macintosh.

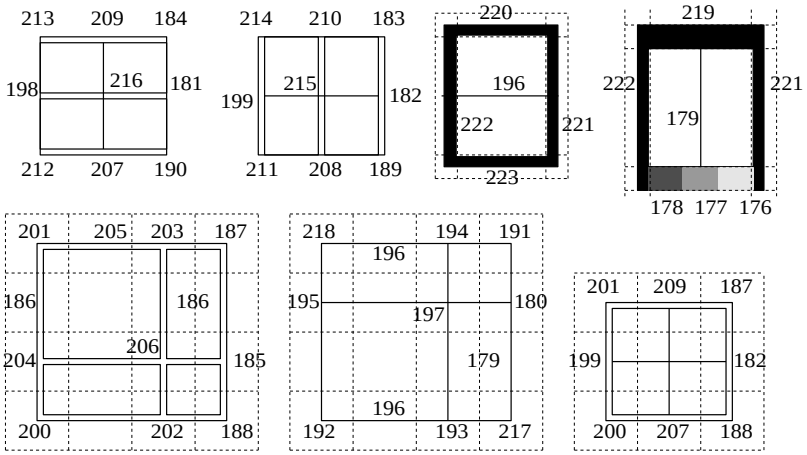


Figure 6-2: Caractères semi-graphiques IBM (les pointillés marquent la limite des caractères).

4 - KILO-, MEGA-, GIGA-OCTET

Ces entités, multiples de l'octet, ne sont pas vraiment des objets informatiques, mais plutôt des unités servant à mesurer leur grandeur et spécialement celle des fichiers, des programmes ou celle des supports les contenant (mémoires, disques, ...). Il faut remarquer que les quantités kilooctet, mégaoctet, gigaoctet sont égales respectivement à 1024, 1024^{2} (1 048 576) et 1024^{3} *octets* et non à 10^{3}, 10^{6} et 10^{9}, valeurs de telles unités en toute rigueur scientifique. C'est donc par abus de langage qu'on les a ainsi dénommées, mais l'erreur commise n'est que de quelques pour-cent. On doit se souvenir que le kilo-octet est exactement égal à 2^{10} octets, le méga-octet à 2^{20}, ..., ce qui constitue des points de repères précieux dans l'évaluation des nombres binaires élevés **(2)**.

5 - LE MOT (ou ENTIER)

Le vocable *mot (word)* désigne généralement une quantité d'information d'une taille égale à celle des *registres* de l'unité centrale. Sur de tels objets, les opérations et manipulations seront donc simples et rapides. Mais la taille du mot dépend des machines.

Cependant, dans la majorité des calculateurs, le *mot* correspond à 2 octets (c'est-à-dire 4 quartets ou 16 bits). C'est en effet une entité intéressante car elle peut contenir 2^{16} = 65 536 valeurs = $FFFF_{16}$ + 1 (le zéro), ce qui veut dire que dans un mot on pourra loger tout **nombre entier positif** *(unsigned integer)* inférieur ou égal à 65 535, zéro compris, ce qui suffit dans beaucoup de cas**(3)**.

(2) C'était la réponse à donner dans l'exercice 3 du chap. II.
(3) Détail qui a parfois de l'importance : les deux octets composant un mot sont rangés en mémoire selon un ordre dépendant du constructeur de l'UC ; par exemple *poids faible poids fort* pour Intel (et les IBM-PC), l'inverse pour Motorola (et les Macintosh).

6 - LES NOMBRES NEGATIFS

Bien que, pendant des siècles, l'humanité se soit passé des nombres négatifs, leur traitement par les calculateurs s'est vite imposé, d'autant plus qu'on a décidé de ramener les soustractions à l'addition du nombre négatif adéquat **(4)**. C'est dans le but de rendre la soustraction correcte qu'on a adopté, pour représenter l'opposé d'un nombre positif, la méthode ci-après, dite **de complément à 2 : on inverse tous les bits du nombre positif, puis on ajoute 1 au résultat**. En effet, l'addition bit à bit d'un nombre et de son inverse binaire, n'ayant affaire qu'à des couples **1+0** ou **0+1**, ne fournira que des **1** sans retenue, c.-à-d. la valeur FFFF pour des mots de deux octets. En lui ajoutant **1**, on obtiendra $10000_{\mathbf{16}}$; si l'on convient de laisser tomber le bit **1** excédentaire à gauche, on obtient **0** pour résultat de l'addition entre un nombre entier et son opposé ainsi défini, ce qui était le but recherché.

Le *mot* informatique de base (souvent symbolisé par *w*) pourra donc contenir, au gré du programmeur, soit un entier positif (ou plus généralement : sans signe) inférieur ou égal à 65 635, soit un entier *signé* pouvant varier équitablement de part et d'autre du zéro, c'est-à-dire de –32 768 à 32 767.

L'informaticien devra connaître par cœur l'ordre de grandeur de ces quantités.

Voici les représentations en machine (ligne inférieure) de quelques entiers *signés* (ligne supérieure) :

–32768	–32767	–2	–1
8000	8001	FFFE	FFFF

(–0)	0	1	2	32767
(1)0000	0000	0001	0002	7FFF

On remarquera que les nombres négatifs, tous de code-machine supérieur ou égal à $8000_{\mathbf{16}}$, comportent bien en tête un bit égal à 1, ce qui les caractérise. On s'exercera à additionner les nombres deux à deux opposés donnés en exemple ci-dessus, pour vérifier que leur somme est bien égale à $10\,000_{\mathbf{16}}$, ce qui respecte la convention adoptée.

Noter que si l'on ajoute 1 à 32 767, on obtient en général –32 768, car 7FFF+1= $8000_{\mathbf{16}}$. L'erreur est donc de taille ; mais elle est classique : la plupart des ordinateurs la commettent sans sourciller. Toute opération qui aurait pour résultat un nombre hors du domaine permis donne en réalité un nombre complètement faux. Le programmeur devra donc toujours être attentif au respect des limites imposées aux entiers.

$0 \le w \le 65\,535$	mot non signé
$-32\,768 \le w \le 32\,767$	mot signé

7 - LE DOUBLE MOT (ou ENTIER LONG)

Les limites ci-dessus s'avérant vite insuffisantes, on a défini une autre entité appelée *double mot* ou *entier long* (souvent notée *D*) et formée de **quatre octets** consécutifs, soit 32 bits. La capacité de cet objet est alors de $2^{\mathbf{32}}$, c'est-à-dire 4 294 967 296. De même qu'avec le mot simple et avec les mêmes conventions, il est possible de donner des valeurs négatives au double mot qui peut alors représenter tous les entiers *D* compris entre les limites $-L$ et $L-1$ en passant par 0 :

Les **adresses** des objets dans la mémoire, souvent appelés *pointeurs*, sont elles-mêmes des objets informatiques, objets en tous points identiques soit au double mot (pointeurs *longs)*, soit au mot simple (pointeurs *courts* ne dépassant pas la taille d'un segment et contenus dans le registre de déplacement, cf. chap. IV, § 2a, p. 27), dans les deux cas non signés.

$0 \le D \le 2L-1$	si *D* non signé	
$-L \le D \le L-1$	si *D* signé,	L = 2 147 483 648.

(4) Vouloir ramener la soustraction $a-b$ à l'addition $a+c$ entre a et l'opposé c de b exige que $a - b = a + c$ et donc que $b + c = 0$.

8 - LES NOMBRES FLOTTANTS (ou REELS)

Nous n'avons parlé jusqu'ici que de nombres entiers. Or les ordinateurs ont été d'abord conçus pour le calcul scientifique et celui-ci utilise largement les **nombres réels**, ceux qui comportent une partie *fractionnaire*. On pourrait convenir de les multiplier par un nombre suffisamment grand pour les transformer en entier, par exemple un million, ce qui permettrait de les *loger* dans les *doubles mots* précédemment définis. Cette technique fastidieuse et très peu satisfaisante a été employée dans les débuts de l'informatique ; elle l'est encore avec des ordinateurs rudimentaires ou dans des cas extrêmes.

La méthode actuellement employée pour représenter les nombres réels reflète la **notation scientifique**. Dans cette notation, un nombre réel s'exprime par le produit d'une **partie significative** (d'où l'on bannit les zéros d'extrémité, lorsqu'ils n'apportent aucune information) et d'un **coefficient** multiple de 10. Ce coefficient s'écrit sous la forme $10^{\mathbf{n}}$.

Le lecteur non initié à cette notation pourra se reporter aux exercices terminant le chapitre I. Cependant, la lecture des sections 8 et 9 ci-après, assez difficiles, n'est indispensable qu'aux personnes intéressées par la programmation scientifique.

La notation scientifique repose sur le caractère fini de la précision des nombres réels, qui découlent en général d'une **mesure approchée**. Peu de mesures courantes dépassent la précision de 10^{-4}, sauf récemment, grâce aux progrès de l'optique cohérente et du laser, pour certaines mesures de longueur (parfois avec plus de 6 chiffres significatifs) et surtout celles des temps (jusqu'à 12 chiffres avec les horloges atomiques). Par exemple, le virus de la rougeole a un diamètre de 0,000 000 18 mètre. Il est beaucoup plus économe, et plus *parlant*, de l'écrire sous la forme 0,18 µm ou 18 10^{-8} mètre, distinguant nettement sa précision (18 unités) de la grandeur de cette unité (10^{-8}) par rapport à celle désignée (le mètre).

Pour représenter de tels nombres dans un espace minimum, celui d'un *double mot* (32 bits), plusieurs procédés ont été inventés. Le plus répandu est celui employé par la famille des *processeurs* Intel 80*x*86 (UC des IBM-PC), répondant à la norme IEEE 754 :

– le zéro s'écrit avec un mot dont tous les bits sont nuls ou dont le premier est égal à 1 (–0). Le zéro a donc deux formes. Hormis ce cas,

– on écrit le nombre réel F en base 2 sous la forme $F = f \times 2^{\mathbf{n}}$, avec la convention suivante, dite de normalisation: $½ < f \leq 1$;

– le signe du nombre (celui de f) est donné par le premier bit ;

– les 7 autres bits du premier octet, plus le premier du deuxième octet servent à représenter l'exposant n, qui pourra prendre $2^{\mathbf{8}} – 1 = 255$ valeurs (la représentation du zéro en occupant une). En fait, à cause de particularités de la représentation (complémentation à 2 pour les exposants négatifs, décalages), n pourra varier de -125 à 128, zéro inclus. La valeur absolue maximale de F sera de $1 \times 2^{\mathbf{128}} = 3{,}40282355\ 10^{\mathbf{38}}$ et sa valeur minimale $0{,}5 \times 2^{\mathbf{-125}} = 1{,}1754945.10^{\mathbf{-38}}$;

– les 23 bits restants serviront à écrire la mantisse f. f étant fractionnaire, chacun de ses bits vaudra $2^{-\boldsymbol{n}}$, si n est son rang compté à partir de la gauche : 1/2, puis 1/4, 1/16 ... etc. Sa valeur absolue pourra varier de 1 à $2^{\mathbf{-23}} \approx 1{,}2\ 10^{\mathbf{-7}}$ par pas de $2^{\mathbf{-23}}$. La précision n'est donc pas tout à fait $10^{\mathbf{-7}}$, mais il s'en faut de peu.

Cette représentation est appelée **flottante** ou *à virgule flottante*, car tout se passe comme si on faisait *glisser* la virgule le long des chiffres pour l'adapter au nombre **(5)**. Pour résumer, la gamme d'excursion des nombres flottants (simples) est la suivante :

$3{,}4\ 10^{\mathbf{38}} > F > 1.18\ 10^{\mathbf{-38}}$	avec 6 à
$-1{,}18\ 10^{\mathbf{-38}} > F > -3{,}4\ 10^{\mathbf{38}}$	7 décimales

9 - LE QUADRUPLE MOT (DOUBLE PRECISION)

Les limitations ci-dessus étant trop restrictives dans certains calculs (par exemple s'il y a beaucoup de **soustractions** entre nombres voisins très grands, ce qui fait chuter la précision), on a défini une nouvelle entité, le *quadruple mot*, qui compte 8 octets. Il est surtout utilisé pour représenter un nombre flottant en un mode appelé **double précision (6)**.

Il réserve encore les deux premiers bits aux signes (signe de f en premier), puis consacre 10 bits à transcrire l'exposant en valeur absolue, qui pourra atteindre ainsi $2^{\mathbf{10}} = 1024$. La valeur absolue du nombre D (pour *double*) pourra donc atteindre $2^{\mathbf{1024}} = 1{,}7977...10^{\mathbf{308}}$ dans les limites suivantes :

$$2{,}2\ 10^{\mathbf{-308}} < |D| < 1{,}79\ 10^{\mathbf{308}}$$
avec 15 à 16 décimales

(5) Voici la représentation de quelques réels : 0000 0000 et 8000 0000 pour 0 et –0, 3F80 0000 pour 1, BF80 0000 pour –1, 4000 0000 pour 2, 3F00 0000 pour 0.5, 7F7F FFFF et 0080 0001 respectivement pour le plus grand et le plus petit réel positif.

(6) Il peut coder un entier d'excursion $\pm 9{,}2\ 10^{\mathbf{18}}$ environ.

Ces nombres sont bien sûr *signés*. Les bornes indiquées sont approchées, mais en deçà des valeurs exactes (les inégalités des deux tableaux précédents sont exactes). On pourra s'exercer à retrouver le détail du mode de représentation de ces objets, calqué sur celui des réels en simple précision.

On aura remarqué que la représentation des flottants, en simple ou double précision, **exclut trois domaines**, dont deux concernent les très grands nombres (on s'y attendait un peu). L'exclusion des grands nombres n'a pas de conséquences trop gênantes. Si un nombre tombe dans l'un de ces deux domaines, il n'est plus considéré comme tel et ne peut plus entrer dans un calcul. Tous les résultats d'opérations dont il devrait être opérande sont signalés comme ±INF (infinis), c'est-à-dire *dépassant les possibilités du calculateur*. C'est beaucoup mieux ainsi.

Il n'en va pas de même avec la troisième zone interdite, celle située près du zéro, mais excluant le zéro lui-même. Quand un résultat de calcul tombe dans ce **domaine interdit**, certaines machines s'arrêtent pour cause de *dépassement inférieur de capacité*, mais la plupart remplacent purement et simplement la valeur interdite par un vrai zéro. Ainsi, le zéro (valeur autorisée) se comporte comme une sorte d'*attracteur* pour ces faibles valeurs interdites.

On peut dire qu'après incursion dans ce domaine interdit, le nombre flottant continuera d'exister, mais **avec une valeur nulle**. Il pourra servir à nouveau d'opérande dans un calcul ! Physiciens et mathématiciens devront se méfier de ce comportement, des valeurs intermédiaires dans les calculs pouvant ainsi se faire piéger et fausser complètement le résultat. Pour y remédier, il faut étudier soigneusement son algorithme, de façon à ne pas provoquer de résultats intermédiaires situés dans la zone interdite. Cette situation anormale sera peut-être corrigée dans l'avenir par les fabricants des circuits de calcul. En attendant, le schéma de la figure 6-3 permettra de mieux se souvenir de cette difficulté.

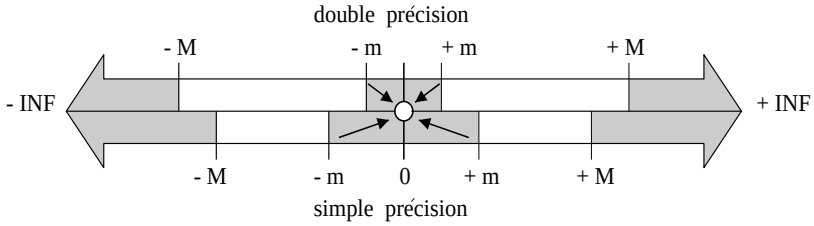


Fig. 6-3: Domaines de validité des nombres réels normalisés,
en simple précision :
$m = 1{,}18\ 10^{-38}$, $M = 3{,}4\ 10^{38}$
en double précision :
$m = 2{,}2\ 10^{-308}$, $M = 1{,}79\ 10^{308}$

10 - LES NOMBRES BCD

La représentation BCD des nombres (surtout des **entiers**) résulte d'un compromis entre le système binaire et le système décimal. Un nombre BCD *(binary coded decimal)* s'écrit comme dans le système décimal, donc avec les dix symboles bien connus : 0, 1,... 8 et 9. **Il est à base 10**, mais chacun de ses chiffres occupe un quartet et s'exprime en binaire comme les dix premiers chiffres hexadécimaux. Les six dernières possibilités du quartet ne sont pas exploitées ; elles sont même appelées *codes interdits*. On verra dans le chapitre VII que cette représentation entraîne quelques difficultés lors des opérations arithmétiques, qui seront toujours plus longues que celles portant sur des *nombres binaires purs*. Voici en exemple les représentations du nombre décimal 235:

0010 0011 0101 en BCD
(on reconnaît les chiffres 2, 3, 5)
1110 1011 en *binaire pur*, (= EB_{16} 14×16+11)

Ces nombres sont employés dans diverses circonstances. Les constructeurs d'appareils électroniques les préfèrent à tous les autres, car ils sont faciles à décoder avant impression ou avant affichage décimal. Dans toute machine où l'on décode beaucoup plus qu'on ne calcule, il est avantageux d'utiliser le système BCD. Ainsi, les compteurs électroniques comptent en BCD (7), les *appareils de mesure* à affichage numérique codent leurs valeurs dans ce système. Il en va de même pour les *calculettes*, parce qu'elles affichent chaque résultat intermédiaire.

Le Basic, conçu d'abord pour des machines de ce type, utilisait au début exclusivement les BCD et il les emploie encore beaucoup, malgré le gaspillage de mémoire (6 symboles sur 16 non utilisés) et la lenteur des calculs qu'il entraîne. En Basic, le BCD occupe 8 octets et représente soit un **entier** de près de 16 chiffres (un bit est utilisé par le signe), soit un **réel à virgule fixe** (sa position est fixée à la compilation par une directive), soit un **réel à virgule flottante** d'excursion voisine de $\pm 10^{\pm 64}$ avec 15 chiffres significatifs (15 quartets, le seizième fournissant les deux signes – 2 bits – et l'exposant – 6 bits –, valeur maximum : 64). En **Cobol**, le nombre BCD entier occupe 10 octets pour fournir 19 chiffres signés. Il y est très employé, surtout pour les calculs financiers qui s'accommodent mal de l'imprécision des flottants. Enfin, certains **logiciels mathématiques** emploient des BCD de longueur arbitraire au gré de l'utilisateur (on parle alors de calcul en **précision infinie**).

(7) C'est la justification du 3e exercice du chapitre III.

11 - LES CHAINES

On appelle chaîne *(string)* une suite de caractères constituant une entité logique. Ainsi, les suites de caractères de l'écriture courante – les mots, les phrases, les paragraphes, les pages... – sont autant de *chaînes* pour l'informatique.

Dans certains calculateurs de la précédente décennie, les chaînes étaient parfois limitées à 8 octets [8] ou à 7. Dans ces dernières, un caractère spécial placé en tête occupait le huitième octet et avertissait l'UC que le mot engagé était une chaîne et non une variable numérique.

Actuellement, les chaînes ont une **longueur quelconque dans des limites données**. Avec certains langages informatiques, tels les premières versions du *Fortran*, cette longueur n'était connue que du seul programmeur qui devait alors ajuster lui-même la portée des instructions manipulant les chaînes. Dans la plupart des langages modernes, un procédé automatique épargne ce souci. Par exemple, en C, la chaîne se termine par le caractère ASCII *null*, de numéro 0. En Pascal ou en Basic, le premier caractère contient la longueur de la chaîne (qui est donc limitée de ce fait à 255 caractères). En langage machine, les deux procédés coexistent. Le programmeur n'a plus besoin alors de trop se préoccuper de la longueur réellement occupée par le texte placé dans ladite chaîne (à condition toutefois d'avoir réservé en mémoire suffisamment de place pour elle) puisque les fonctions de traitement de chaînes ont un moyen pour délimiter leur objet.

12 - LES OBJETS COMPOSITES

Les objets composites sont des ensembles formés par les objets élémentaires développés ci-dessus, mais manipulables en bloc, **sous un même nom**. La **chaîne**, par exemple, est un objet composite formé uniquement de caractères.

Les autres objets composites sont les **tableaux** *(arrays)*, formés d'objets quelconques, mais tous identiques entre eux, et les **structures**, formées quant à elles d'objets disparates, chaînes, entiers, flottants, ...

De tels objets ne donnent pas lieu à une représentation spéciale en mémoire. A vrai dire, l'unité centrale ne les reconnaît même pas comme tels. Aussi nous n'en dirons guère plus dans ce chapitre axé sur la machine, les réservant pour les chapitres concernant les *langages évolués* (chap. IX, §5 et X, §1 et 2). Les tableaux sont connus depuis les débuts de l'ère des calculateurs et n'ont que très peu évolué depuis. Les structures, quant à elles, sont en plein développement : elles sont devenues l'objet le plus général de l'informatique. Leur emploi, déjà très important en gestion (Cobol), facilite la manipulation des *concepts*, matériau de prédilection pour les langages orientés vers l'intelligence artificielle.

13 - LES INSTRUCTIONS

Ce sont des suites d'octets comprenant des chaînes et généralement des nombres. Dans une instruction, l'une des chaînes au moins joue un rôle prépondérant : appelée **code-instruction**, **mot-clé**, **mot réservé** ... , selon le langage utilisé, c'est elle qui indique à l'UC le travail demandé. Elle se comporte un peu comme le *verbe* dans le langage humain. Les autres chaînes ou les nombres complétant l'instruction constituent plutôt des **paramètres** définissant les *objets* sur lesquels portera le travail.

Les instructions peuvent être écrites directement en **langage machine** (en abrégé, **LM**), c'est-à-dire sous forme de nombres hexadécimaux (ou de bits). Cette écriture fastidieuse est presque toujours remplacée par le langage machine **symbolique** (souvent appelé *assembleur*), qui est un peu moins hermétique (on en reparlera dans le chapitre suivant). Mais la grande majorité des programmes est écrite dans des **langages évolués** (chapitres IX et X) dont les instructions, souvent assez longues, sont nettement plus proches du langage humain et beaucoup plus faciles à lire et écrire que celles en langage machine. Cependant, l'instruction numérique étant la seule comprise par le décodeur de l'UC, toute autre instruction devra être traduite sous cette forme avant exécution.

(8) Certains calculateurs simples ne peuvent manipuler que des objets de 8 octets : ceux-ci peuvent alors contenir soit une chaîne, soit un BCD, soit un flottant en double précision (d'où simplicité, mais gaspillage de mémoire).

14 - PROGRAMMES ET SOUS-PROGRAMMES

Un programme est une **suite d'instructions** décrivant en détail les opérations et les manipulations à faire exécuter par la machine pour parvenir au résultat escompté. Ses instructions, sauf contrordre, s'exécuteront *en séquence*, c'est-à-dire dans l'ordre où elles ont été écrites.

Un programme (*routine*) commence toujours par une sorte de **préambule**, dans lequel le programmeur **définit** les objets utilisés, surtout les objets composites, afin de leur *réserver* la place adéquate en mémoire. Il donne également au début différentes informations, dont le nom des fichiers dans lesquels, le cas échéant, l'UC devra aller rechercher d'autres éléments.

On peut même à ce niveau définir des chaînes spéciales qui constitueront autant de nouveaux mots-clés ou de véritables petits sous-programmes qu'on appelle **macro-instructions**. Cette façon de faire permet d'éviter d'écrire de nombreuses fois des séquences d'instructions identiques.

C'est après ce préambule indispensable que commence la série des instructions dites *exécutables*.

Un **sous-programme** *(subroutine)* est un ensemble d'instructions qui, à l'instar d'un programme, possède une bonne **homogénéité**, mais pas d'autonomie propre. On peut dire qu'il décrit le détail d'une *sous-tâche* à accomplir, surtout si, au cours du travail demandé, il est souvent fait appel à cette sous-tâche. Pour pouvoir être exécuté, un sous-programme doit toujours être *appelé* par le **programme principal** (*main*). Mais il pourra l'être de nombreuses fois ou bien être appelé par un autre sous-programme. Mieux encore, avec certains langages, un sous-programme peut s'appeler lui-même : on appelle cette faculté la *récursivité* (autant prévenir immédiatement qu'on doit alors prendre quelques précautions si l'on ne veut pas qu'un tel programme s'appelle indéfiniment...!).

Le concept de sous-programme permet de diviser l'ensemble d'un programme en plusieurs **modules** dont la compréhension et la mise au point sont beaucoup plus faciles que celles d'un énorme bloc. Le sous-programme possède suffisamment d'indépendance pour lui permettre d'être écrit par un programmeur autre que celui chargé du programme principal. Cette notion de **modularité** autorise donc la répartition d'un travail important entre plusieurs programmeurs ou bien l'intégration à un programme nouveau d'une sous-tâche précédemment mise au point. Tous les langages de programmation n'ont pas cette possibilité. C'est en particulier son absence dans le Basic ordinaire qui lui a valu la majorité des reproches dont il a fait l'objet.

15 - LES FICHIERS

Un programme possède une *unité logique*. Mais il faudra bien l'écrire quelque part dans une *mémoire de masse*, sur un *support physique*, pour en assurer la conservation. Pour le retrouver aisément, on le placera dans un sous-ensemble de ce support, un *fichier*.

L'origine du mot est à rechercher dans les paquets de *cartes* sur lesquelles, jusque vers 1975, on écrivait les instructions à raison d'une par carte. Ces cartes, ou *fiches*, une fois groupées, remplissaient une boîte, un tiroir ou un *fichier (file* en anglais).

Les fichiers peuvent contenir autre chose que des programmes, par exemple des *données*. Fichiers et programmes ne sont donc pas des notions équivalentes, d'autant plus qu'un fichier peut parfaitement contenir plusieurs sous-programmes ou même, exceptionnellement, plusieurs programmes. Plus proches de la machine que les programmes, les fichiers et leur manipulation par le *système d'exploitation* seront décrits dans le chapitre VIII.

16 - LE LOGICIEL

Un logiciel *(software)* peut être défini comme un **ensemble** de fichiers, de programmes ou de sous-programmes écrits pour atteindre un objectif donné d'assez grande envergure. Un logiciel présente donc une grande **cohérence**. Nécessitant parfois plusieurs *années-hommes* de mise au point, il peut être écrit en même temps par plusieurs *programmeurs* qui doivent s'imposer une discipline très stricte pour obtenir une parfaite coordination de leur travail.

Les divers fichiers d'un logiciel sont très peu autonomes. La suppression ou la perte de l'un d'eux bloque souvent le fonctionnement de l'ensemble ; ils font leur la devise célèbre : *tous pour un, un pour tous*.

Les travaux de bureautique se réalisent avec l'aide de logiciels presque toujours achetés dans le commerce. Ils sont quelquefois appelés *progiciels* ou *applications*. Un *traitement de texte* par exemple est un logiciel. S'ils sont produits par une grande firme, les logiciels évoluent dans le temps et s'adaptent aux différentes générations de machines de la même famille, ce qui donne lieu à la commercialisation de *versions* successives *(releases)*, dont le perfectionnement va en croissant ; il est souvent possible d'accéder

au niveau de la dernière version, lorsqu'on en possède une plus ancienne, en achetant seulement une *mise à jour (up to date)*.

Notez que de tels ensembles de fichiers s'appellent des logiciels, mais que l'*ensemble des logiciels*, programmes isolés, fichiers ... implantés dans les mémoires d'un ordinateur s'appelle tout simplement **le** logiciel. Il y a peut-être là source à confusion, regrettable en informatique, bien qu'assez répandue dans le langage courant, où l'on assimile souvent le tout à la partie (cf. *transistor*, *caravane*...). Les Anglo-Saxons apparemment distinguent mieux les deux concepts : *software* pour le logiciel en général, *application* pour un logiciel donné (que certains informaticiens et technocrates appellent des *codes*, pour compliquer encore un peu plus !).

17 - HIERARCHIE LOGICIELLE

Sans vouloir trop rigidifier les concepts décrits dans ce chapitre, on peut dresser le tableau ci-contre qui aura le mérite de fournir au novice une image lui rappelant, en première approche, la hiérarchie des objets manipulés par l'informatique.

On rappelle que tous ces objets sont formés de *bits*, dont le nom coiffe la pyramide. Le bit donne naissance à l'*octet* et au *caractère*, termes en général équivalents. A droite, figure une colonne à valeur purement indicative concernant plutôt la mesure en octets du logiciel et de ses constituants. La colonne de gauche a rapport au texte (*chaînes*), les colonnes centrales aux *nombres*, simples, puis disposés en *tableaux*. Texte et nombres se composent pour donner des *structures* ou bien des *instructions* (on pourrait d'ailleurs regarder ces dernières comme des structures). A la base de la pyramide, se situe **le** *logiciel* de l'installation, formé lui-même de *plusieurs logiciels*, eux-mêmes comprenant de nombreux *fichiers*. Ces derniers sont le support physique des *programmes* et des *sous-programmes*.

On signalera pour clore ce chapitre que la liste des objets cités n'est pas exhaustive. Certains langages en emploient d'autres. Il ne faut donc pas considérer ces concepts comme trop rigides. A partir de l'octet et en langage machine, tout concepteur peut définir les mots qu'il veut, bien que les unités centrales soient agencées pour faciliter les opérations sur l'un des types décrits précédemment. On aura noté l'importance des nombres **réels** et peut-être remarqué qu'ils peuvent exister sous deux formes : à *virgule fixe* (forme à vrai dire peu employée sauf en Cobol, cf. page 47) et à *virgule flottante*, c'est-à-dire avec un exposant variable. Les mots *flottant* et *réel* ne sont donc pas synonymes, bien qu'en pratique souvent confondus. Signalons enfin que certaines unités de calcul travaillent avec plus de chiffres que ceux exigés par les normes : ainsi les *coprocesseurs* arithmétiques Intel et compatibles effectuent tous leurs calculs avec des nombres de 10 octets. Avec certains langages, le programmeur peut utiliser ces nombres dits de **précision étendue** (18 chiffres significatifs, excursion $\pm 10^{\pm 4932}$).

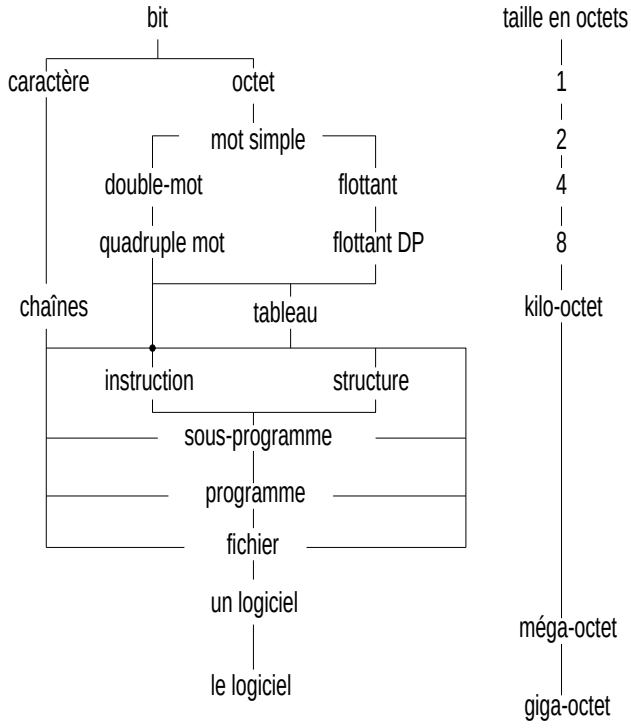


Fig. 6-4 : la hiérarchie logicielle.

EXERCICES

Le lecteur qui situe mal la taille des nombres employés par l'informatique est invité à résoudre les exercices ci-après afin de prendre des repères par rapport au concret, c'est-à-dire au monde physique.

1 - En comptant le nombre de caractères contenus dans une ligne de texte et celui des lignes contenues dans une page, estimez approximativement le nombre de caractères de ce manuel. Sachant qu'une disquette peut contenir 1,44 méga-octet, combien de telles disquettes sont-elles nécessaires pour conserver tout le texte (hors figures) de ce livre ?

2 - Même question pour l'*Encyclopædia Universalis* qui compte 20 volumes de 1100 pages environ, chaque page contenant 3 colonnes de 100 lignes à raison de 44 caractères par ligne. Pour conserver ce texte (toujours hors figures), combien de disquettes, combien de disques optiques de 800 mégaoctets ?

3 - Combien de générations approximativement se sont-elles succédé sur Terre depuis les premières manifestations de l'homme, il y a environ 2,5 millions d'années ? Ce nombre peut-il *tenir* dans un mot simple, un mot double ?

4 - L'étoile α du Centaure, située à environ 4,3 années-lumière, est la plus proche de la Terre. Une année-lumière est une unité de longueur égale à la distance parcourue en un an dans le vide par la lumière, à raison de 300 000 km/s. Combien de femtosecondes la lumière de cette étoile met-elle pour parvenir jusqu'à nous ? Combien y-a-t-il de femtomètres entre elle et nous ? Quels sont les objets informatiques nécessaires pour contenir ces nombres ? On remarquera que le premier nombre cité ne comportant que 2 chiffres significatifs, il serait incorrect de donner un résultat avec plus de 2 chiffres différents de zéro, c'est-à-dire avec une précision meilleure que 1% (et même 2%).

5 - La plus grande durée connue est l'âge de l'Univers, environ 15 milliards d'années, et la plus petite (pour l'instant) la femtoseconde. Faites le rapport des deux. Ce nombre peut-il être *logé* dans un flottant simple ? (c'est pourtant le plus grand nombre qui puisse intervenir dans une mesure de temps).

6 - La masse de la Terre est environ 332 300 fois inférieure à de celle du Soleil, laquelle est de $1{,}99\ 10^{30}$ kg. En admettant que la Terre soit formée uniquement de silicium, dont l'atome a une masse de $4{,}7.10^{-23}$ gramme, combien y-a-t-il d'atomes sur Terre [9] ? Ce nombre peut-il être contenu dans un flottant simple ? Pourriez-vous dire pourquoi ce nombre est bien supérieur aux résultats des exercices 3, 4 et 5 ?

7 - Supposons qu'un chimiste ait trouvé un moyen rentable d'extraire l'or de l'eau de mer. Calculez la masse d'or qu'il peut espérer récupérer, le volume des océans étant de 1370 millions de kilomètres cubes, leur teneur moyenne en or de 4 microgrammes par tonne, la masse spécifique de cette eau de 1,035 g/l. Avec quels types d'objets informatiques pourrait-on mener à bien ces calculs ? Comment s'y prendrait-on si on ne disposait que d'entiers *courts* [10].

(9) Le nombre de particules contenues dans l'Univers observable serait d'environ 10^{80}.
(10) Un peu comme autrefois avec les règles à calcul. Expliquez.

Chapitre VII

LE LANGAGE MACHINE

Tout comme les langues humaines, un langage informatique comprend des règles et un vocabulaire : il permet d'écrire des instructions décrivant au calculateur comment utiliser et traiter les objets étudiés dans le chapitre précédent.

Les langages informatiques sont assez nombreux, mais il n'y a qu'un seul **langage machine** pour un calculateur donné. Contrairement aux autres, le langage machine est défini par le constructeur même de la puce qui constitue l'unité centrale *(processeur)*. Il va sans dire qu'il varie d'un constructeur à l'autre, mais, malgré cette variété, il est possible d'en discerner des traits communs. De plus, au sein d'une même famille de *processeurs*, telle celle des 80*x*86 (qui comprend les 8080, 8088, 80286, 80386, 80486, Pentium), le langage manifeste une compatibilité suffisante pour être accepté par tous les circuits postérieurs à celui pour lequel il a été écrit. En fait, d'une version à l'autre, chez un même constructeur, on assiste à un enrichissement du langage plutôt qu'à sa redéfinition.

Dans ce chapitre, on se propose de survoler quelques aspects généraux des **langages machine** (**LM** en abrégé). Sous ce vocable, on a réuni ce que certains considèrent comme deux langages différents : le LM *codé* ou *numérique* et le LM *symbolique* (communément appelé *assembleur*). Nous les considérerons comme deux facettes de la même réalité, beaucoup trop parallèles pour qu'on en dissocie l'étude.

1 - LE LANGAGE NUMERIQUE

C'est un langage, formé uniquement de **nombres**, dans lequel sont écrits *en définitive*, par un moyen ou par un autre, les instructions et les programmes *chargés* en mémoire et exécutables par le calculateur. Chaque instruction possède une adresse, celle où elle sera mise en mémoire. L'instruction est formée d'un *code opératoire*, souvent de la taille d'un octet, puis d'un ou de quelques *opérandes*. La longueur de chaque instruction est généralement variable **(1)**, mais à chaque *code* correspond une longueur précise ; c'est en reconnaissant ce code que l'UC détermine la longueur exacte de la chaîne d'octets à charger dans le décodeur et en déduit l'adresse de l'instruction suivante.

Il existe un **code opératoire** pour chaque fonction élémentaire permise par le *processeur*. Les principales sont : la *recopie* du contenu d'un opérande dans un autre, les *opérations arithmétiques* fondamentales, la *comparaison* entre deux objets, le *déroutement* du programme, l'*émission* d'un octet vers les portes de sortie, la *réception* d'un autre sur celles d'entrée, etc.

L'**opérande**, quant à lui, peut être soit un *nombre pur*, soit le numéro de l'un des *registres* centraux, soit une *adresse* de la mémoire, soit celle d'une *porte* d'entrée-sortie, soit celle d'une autre instruction.

Lorsque le code opératoire correspond exactement à un octet, comme dans le langage des premiers calculateurs, un programmeur exercé parvient à comprendre ces instructions sans trop de difficultés. Mais actuellement, les codes sont devenus très complexes, les opérandes étant par exemple "imbriqués" avec les codes opératoires au sein d'un même octet, qu'il faut alors interpréter bit par bit. Ce travail, tout à fait accessible à la machine, devient trop ardu pour l'être humain. Aussi, après l'époque héroïque des premiers calculateurs, personne, à quelques exceptions près, n'écrit plus dans ce langage.

Parmi les exceptions, citons-en trois importantes : tout d'abord, il arrive qu'un programmeur veuille examiner ou même modifier directement un détail du langage implanté en mémoire, car c'est là que réside la précision maximum. Pour le faire, il lui faut connaître un minimum de langage machine numérique.

Deuxièmement, on a parfois la possibilité d'introduire directement du langage machine (en hexadécimal pur) dans un programme écrit en langage évolué (c'est le cas du C, du Pascal...) ; cela peut s'avérer nécessaire pour des manipulations d'objets inaccessibles aux langages de haut niveau. Il serait dommage que les programmeurs confirmés se privent de cet atout par ignorance totale du LM.

La troisième exception est de taille, puisqu'il faut bien qu'une équipe se dévoue au départ pour écrire un logiciel capable de **traduire** en hexadécimal des instructions écrites dans un langage moins difficile. Ce logiciel ne peut être écrit lui-même qu'en hexadécimal et au sein de l'équipe qui conçoit le *processeur*.

(1) Sauf dans le contexte de l'architecture RISC, mentionnée en fin de chapitre.

2 - LE LANGAGE SYMBOLIQUE (ASSEMBLEUR)

Le langage symbolique constitue la première étape imaginée pour se dégager du jargon rébarbatif de la machine. Il ne diffère cependant que très peu du précédent et seulement par le vocabulaire. La structure des instructions est rigoureusement la même. Les nombres hexadécimaux représentant codes opératoires ou objets sont remplacés par des *expressions littérales*, c'est-à-dire par des **mots** beaucoup plus faciles à retenir. Le *nombre* cède le pas à la *lettre*. Enfin, les nombres purs pourront s'exprimer maintenant dans le système décimal.

a – les adresses des instructions du langage numérique sont remplacées – et seulement si elles sont nécessaires – par des noms (ou *littéraux*) appelés *étiquettes (labels)*. Ces noms sont choisis par le programmeur en relation avec le contexte. Comme dans la plupart des langages évolués, on n'explicite les étiquettes que lorsque c'est indispensable, c.-à-d. pour désigner une instruction inaugurant une séquence spéciale, à laquelle on aura besoin de se référer. Toutes les autres adresses sont générées par la machine sous forme numérique à l'insu du programmeur.

b – les codes opératoires sont eux aussi remplacés par des *littéraux* courts (des mots, ou plutôt des abréviations), les **mnémoniques** : ces mnémoniques, formés de 2 à 4 lettres, sont, à la différence des étiquettes, **imposés** par le langage. Le mnémonique le plus important est celui relatif à la *recopie* : il est souvent symbolisé par le mot **MOV** (de l'anglais *move*, déplacer). Vient ensuite celui de l'addition (**ADD**). Citons quelques codes opératoires classiques : **LOA** pour *charger (load)*, **STO** pour *mémoriser (store)*, **SUB** pour soustraire, **CMP** pour comparer, **JMP** pour *aller à* (de *jump*, sauter), **IN** pour recevoir une information d'un périphérique, **OUT** pour l'émettre ... A tous ces mnémoniques du langage symbolique, il correspond un code hexadécimal ou binaire dans le langage machine numérique (parfois ce code est mélangé, comme on l'a déjà dit, au code de l'opérande).

c – les opérandes du langage symbolique ne sont plus systématiquement des nombres. Les plus usités, les registres généraux, sont désignés par des lettres comme **A**, **B**, **C**, **D**, **E** [(2)]... On a fait remarquer que ces registres pouvaient être décomposés en deux pour traiter des *mots* de longueur égale à la moitié de leur capacité nominale (par exemple des octets dans des registres de 16 bits). Dans ce cas, on utilise les suffixes **X**, **L** ou **H** pour désigner respectivement la totalité, la partie basse (*low*) et la partie haute de ces registres. Par exemple, le registre **A** s'appellera **AX**, sa partie de faible poids **AL** et sa partie haute **AH**.

Les adresses des autres opérandes sont remplacées par des noms suggestifs, laissés – moyennant quelques règles – au libre choix du programmeur. Quant aux nombres purs, on peut les écrire sous les trois formes, hexadécimale, octale et décimale. Signalons que les nombres purs s'appellent en langage machine des *constantes immédiates* et que, pour les distinguer sans ambiguïté des adresses, les mnémoniques les utilisant se terminent souvent par la lettre **I** (*immédiat*).

3 - LES RUPTURES DE SEQUENCE

Les instructions du programme sont exécutées *en séquence*, c.-à-d. dans leur ordre d'écriture. Ce procédé rigide ne peut pas traduire la complexité de la majorité des problèmes. Par exemple, le programme qui calcule les racines d'une équation du second degré doit exécuter des séquences différentes selon que le discriminant Δ a une valeur positive, négative ou nulle. De même, celui qui calculerait l'impôt sur le revenu doit savoir s'adapter aux multiples situations des contribuables.

Pour faire face à cette diversité, le langage machine dispose d'une instruction, le **saut**, qui lui permet de *sauter* une partie du programme. Il existe deux grands types de sauts : dans le premier type, le saut ne dépend d'aucune condition et s'appelle **saut inconditionnel** (en général symbolisé par le code **JMP**, *jump)* : on saute (comme dans le jeu de l'oie) directement à l'*étiquette* désignée et on *déroule* le programme à partir de là. Tout aussi importantes sont les instructions du second type, celles de **saut conditionnel**, qui permettent à l'UAL, après examen d'une *condition*, de sauter, si elle est remplie, à l'instruction dont l'étiquette figure en opérande, ou, sinon, de continuer le programme en séquence.

En fait, pour décider si elle procède ou non au saut conditionnel, l'UAL examine la valeur de l'un des *indicateurs* du *registre d'état* (chap. IV, p. 26). Ceux-ci *(balises, flags)* sont répartis en indicateurs de *zéro*, de *débordement*, de *retenue*, de *signe*... Il sont *armés* (c.-à-d. portés à la valeur 1) par le résultat des opérations précédentes. Par exemple, dans la résolution de l'équation du second degré, un résultat négatif lors du calcul du discriminant Δ armera l'indicateur *signe*. Une instruction de saut placée immédiatement après permettra le déroutement du programme, lorsque l'indicateur *signe* est armé, vers la séquence correspondant au cas $\Delta<0$.

(2) Autres appellations (selon la machine): r1, r2, r3..., $1, $2, $3, ...

Avant un saut, on peut vouloir effectuer une comparaison. Ainsi,

CMP a, b

pourra dire "comparer *b* à *a*". Automatiquement, l'un des indicateurs reflétera le résultat de la comparaison. Ce sera souvent l'indicateur S, celui de signe, la comparaison n'étant rien d'autre qu'une soustraction. L'instruction de saut qui suit celle de comparaison pourrait s'écrire

JS *étiquette*

sauter à *étiquette* si S est armé, c.-à-d. si le résultat précédent était négatif (si $a < b$), ou bien **JNS** *étiquette* (sauter si S n'est pas armé). Bien entendu, si la condition n'est pas satisfaite, on continue en séquence. D'autres codes de saut sont attachés aux autres indicateurs. Ils constituent, avec l'instruction **CALL** (cf. §11), les seules possibilités d'écart à la loi générale d'exécution *séquentielle* des instructions.

4 - LES CHAMPS DE L'INSTRUCTION

L'instruction symbolique comprend quatre *champs* :

Etiqu. :	Code opérat.	Opérande(s) ;	Comment.

– le premier, facultatif, est celui de l'*étiquette* ; si elle existe, elle doit être suivie d'un signe spécial, pour la séparer du champ suivant, par exemple le caractère *deux-points* (:) ;

– le second champ contient le *code opératoire*, ou *mnémonique*, ou encore *opérateur*, mot de quelques lettres faisant obligatoirement partie de la liste des codes autorisés et dont on a déjà parlé ;

– le troisième contient les *opérandes* éventuels. Leur nombre dépend du code opératoire, mais leur écriture admet une certaine liberté. On a dit que les registres possèdent des noms imposés, mais les cellules-mémoire sont désignées par des noms suggestifs, laissés au gré du programmeur. Ces opérandes sont assimilables aux *variables* du calcul algébrique, mis à part le fait qu'en algèbre, les variables sont désignées par une seule lettre affectée éventuellement d'indices, alors qu'en LM symbolique, le nom des opérandes peut comporter plusieurs lettres ;

– le dernier champ est facultatif. S'il est présent, il doit être séparé des opérandes par le symbole approprié, par exemple par le point-virgule. Ce champ n'est pas traité par la machine et ne constitue qu'un **commentaire** placé là pour expliquer à un lecteur, ou bien pour rappeler à l'auteur lui-même, le rôle joué par l'instruction.

5 - EXEMPLE

On va s'appuyer sur l'exemple suivant pour analyser de plus près le langage machine symbolique. Un programmeur aurait pu écrire la séquence suivante :

```
debut :
   MOV AX, ventes ;      ventes = recette du magasin
   MOV BX, frais ;       frais = frais généraux
   SUB AX, BX
   LOA BX, achats ;      frais d'achats
   SUB AX, BX ;          résultat net dans AX
   MOV resul, AX ;       sauvegarde du résultat
   JNS impot ;           si resul positif, saut à impot
   CALL deficit;         traitement du déficit
   JMP suite
impot :
   MUL AX, taux ;        taux d'imposition en pour-cent
   MOV cent, 100 ;       passage par la variable cent=100
   DIV cent ;            division de AX par cent
   STO impot AX
   ......                ......
suite :  ......          ........
```

Commentons ce petit fragment de programme [3], instruction par instruction : la première ligne, repérée par l'étiquette **debut**, demande le *chargement* dans le registre **AX** du contenu de la cellule-mémoire **ventes**. Noter l'ordre des deux opérandes : **le complément direct est placé <u>après</u> l'indirect** ; en informatique, l'ordre normal est : *opérande cible*, *opérande source*. Se souvenir également que les opérateurs de *transfert* (**MOV** entre autres), malgré leur nom, ne modifient pas le contenu de l'opérande source ; ce sont en réalité des opérateurs de *copie*. Le point-virgule marque la fin de l'instruction : tout ce qui le suit sur la même ligne est ignoré de la machine et constitue un *commentaire*.

La seconde instruction ressemble à la première : on place le contenu de la *variable* **frais** dans **BX**. La troisième soustrait **BX** de **AX**, le résultat étant toujours placé dans la *cible*, ici **AX**. N'étant pas *commentée*, elle n'exige pas de point-virgule. A propos de cette instruction, remarquons l'usage fréquent du registre **AX** : appelé *accumulateur*, c'est lui qui participe le plus aux opérations arithmétiques.

(3) Les omissions d'accent dans ce programme sont volontaires. Les langages informatiques, nettement *anglophiles*, n'acceptent que les caractères de numéro ASCII inférieur à 128, donc pas les lettres accentuées (sauf dans les chaînes et les commentaires). D'autre part, si les étiquettes sont placées ici sur une ligne spéciale, c'est pour gagner en lisibilité.

La quatrième ligne se distingue par le mnémonique **LOA** *(load,* charger dans un registre) qui s'oppose à **STO** *(store,* sauvegarder en mémoire) ; ces deux opérations sont souvent regroupées sous le seul code **MOV** . La sixième emploie **MOV** dans une sauvegarde, c.-à-d. une écriture en mémoire centrale.

La septième emploie le code de *saut conditionnel* **JNS** et signifie *sauter à l'étiquette* **impot** *si l'indicateur de signe* S *n'est pas armé,* c.-à-d. si la dernière opération n'a pas eu un résultat négatif. Le code **MOV** ne modifiant jamais un indicateur, la valeur de S dépend du signe de **AX** après l'instruction **SUB**, donc de **resul**. Le programme continuera en séquence si **resul** est négatif, sinon il se déroutera sur l'instruction marquée par l'étiquette **impot** et continuera en séquence à partir de là.

La huitième instruction est un appel à un sous-programme qui, comme son nom l'indique, traitera le cas du bilan négatif, comme c'est normal, puisque l'instruction **JNS** n'aura alors provoqué aucun saut. Comme nous le verrons plus tard (§11), après l'exécution d'un sous-programme, l'UAL reprend l'exécution du *programme appelant* à l'instruction située juste après le **CALL**. Pour éviter de lui faire dérouler la partie consacrée au cas du résultat positif (étiqueté par **impot**), il faut faire exécuter à cet endroit un *saut inconditionnel* d'une portée telle qu'on évite la partie non concernée.

Si le bilan est positif, on saute à l'étiquette **impot** qui introduit la séquence convenant à ce cas. On y remarque une multiplication (**MUL**) et une division par 100 (**DIV**), qui calculeront le montant de l'impôt. Ces opérations appellent quelques commentaires : on a supposé ici – le cas est fréquent – que le code de division n'acceptait pas d'opérande *immédiat* (un nombre) ; il a donc fallu passer par la variable appelée **cent** à laquelle on affecte justement la valeur 100. De plus, **MUL**, tout comme **DIV** n'accepte que des opérandes *entiers*. On ne peut donc employer un taux d'imposition fractionnaire, ce qui nous a obligé d'abord à multiplier **resul** par un taux 100 fois plus élevé, puis à diviser le grand nombre obtenu par 100 pour aboutir à l'impôt au franc près. Le calcul en francs au centime près aurait exigé l'emploi soit de *nombres flottants*, soit d'entiers exprimant les centimes (cette option est rarement conseillée, car les divisions de type *entier* provoqueraient la perte des parties fractionnaires).

6 - AUTRES CODES IMPORTANTS

Bien d'autres types d'instructions sont généralement disponibles. Nous citerons trois catégories importantes.

a - les instructions de **répétition** qui provoquent la répétition d'une ou d'un groupe d'instructions, aussi longtemps qu'un registre-compteur n'a pas atteint la valeur zéro. Bien sûr, ce cas n'a d'intérêt que si au moins un des opérandes varie. On appelle ce processus **une boucle** *(loop)* : un opérande (ou plusieurs) comporte une adresse composite dépendant en partie d'un registre dit d'*index* qui sera *incrémenté* ou *décrémenté*[(4)] automatiquement à chaque exécution.

La plupart du temps, l'opérande dont l'adresse varie ainsi est un élément d'*objet composite*, *tableau* ou *chaîne*. **La boucle est le moyen idéal pour traiter un tableau**. Ces deux concepts, indissociables, ont grandement contribué à la puissance de l'informatique.

b - les instructions de **décalage** *(shift)* ou de **permutation** *(rotation)* des bits dans un mot, vers la gauche ou vers la droite. Lors des décalages, le bit d'extrémité est expulsé : il *tombe* d'abord dans l'*indicateur de retenue* avant d'être perdu au décalage suivant. Ces manipulations sont nécessaires dans la multiplication et la division (cf. chap. II, pp. 11-12) ainsi que dans la mise en forme des octets formant les messages échangés avec des appareils extérieurs.

c - les instructions d'**ajustement** des opérations sur les **BCD**. Ces nombres, définis dans le chapitre précédent (§10, p. 47), font l'objet d'un traitement spécial. Dans un premier temps, on effectue sur eux, quartet par quartet, les mêmes opérations arithmétiques que sur les *binaires purs* (un binaire pur est le contraire d'un BCD). Puis, sur un ordre explicitement écrit par le programmeur, une correction intervient, consistant à détecter dans le résultat les quartets contenant des **codes interdits** (cf. p. 47) et à leur ajouter 6, ce qui transforme en BCD le résultat hexadécimal. L'UAL doit bien sûr tenir compte des retenues éventuellement provoquées par cette addition corrective.

d - les **appels d'interruptions logicielles**, de la forme **INT** *n* , où *n* est le numéro de l'interruption. Celles-ci sont de petits sous-programmes faisant partie du *système d'exploitation,* écrits en LM (avec des instructions **IN**, **OUT**) pour lire, écrire, organiser les données hors de la mémoire centrale *(fonctions système,* cf. chap. VIII, §1, p. 61).

(4) *Incrémenter (décrémenter)* une variable signifie lui ajouter (retrancher) une unité ou tout autre quantité explicite.

7 - LE CODAGE (ASSEMBLAGE)

Alors qu'un programme en langage numérique est exécutable tel quel par l'ordinateur, un programme en langage symbolique ne peut être exécuté qu'après sa traduction en langage numérique par un logiciel généralement appelé *assembleur*. Ce mot étant tout à fait impropre, nous proposons de l'appeler *codeur*. La figure 7-1 schématise l'opération à effectuer. Ce transcodage est assez simpliste : il se borne à remplacer les symboles littéraux par leur équivalent numérique, sans rien changer ni à l'ordre des instructions, ni à celui des champs (seule exception : le *champ commentaire* est complètement ignoré).

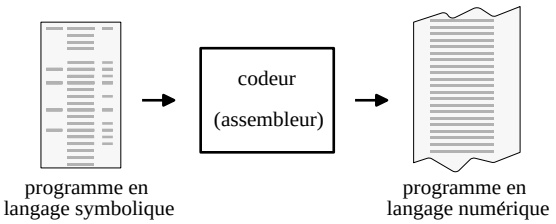


Fig. 7-1 : Codage (assemblage) du programme symbolique.

Pour effectuer cette traduction, le codeur utilise trois dictionnaires ou *lexiques* : le premier contient la liste des *mnémoniques* et, en face, leur équivalent numérique. Le codeur substitue sans difficulté le code au mnémonique. Le deuxième dictionnaire est vide au départ. Le codeur y crée une entrée chaque fois qu'il rencontre une *étiquette* et lui affecte l'adresse de l'instruction à laquelle elle est accolée (cette adresse de définition doit être unique). Une référence à une étiquette pouvant arriver avant sa définition (par ex. dans une instruction de saut en avant), le programme symbolique doit *passer* deux fois dans le codeur pour *résoudre* complètement les symboles d'étiquettes.

Le troisième dictionnaire est d'un emploi plus délicat. Vide également au départ, il fournira l'adresse de chaque *opérande* symbolique. Le codeur remplira ce lexique à mesure qu'il traduit directives et instructions. Quand le codeur rencontre un nom symbolique, il cherche si ce nom figure dans le lexique des mnémoniques, puis dans celui des étiquettes et enfin dans celui des opérandes. S'il ne figure dans aucun, il crée pour lui une *entrée* dans ce dernier et lui alloue une zone mémoire commençant à la première adresse disponible. Pour déterminer la longueur de la zone à attribuer, le codeur doit connaître le type de l'objet en question. Si les directives et déclarations figurant dans le *préambule* (cf. ci-après) sont insuffisantes pour y parvenir, le codeur signalera une erreur.

Quand tous les symboles auront pu être traduits, ce transcodage fournit un programme directement exécutable s'il est court (c.-à-d. ne dépasse pas la longueur d'un segment) et s'il n'est lié à aucun autre programme. Si ce n'est pas le cas, une ou deux étapes ultérieures seront nécessaires *(édition de liens* et *chargement)*. C'est seulement alors que les adresses complètes (segments et déplacements) seront attribuées aux instructions et aux données du programme.

Avant de quitter le codage du langage symbolique en numérique, signalons que l'opération inverse est possible. On peut demander à l'ordinateur de nous traduire (partiellement) en langage symbolique un programme codé en numérique. Seuls les mnémoniques sont restitués. Les étiquettes et les données restent sous forme d'adresses numériques. Les différentes instructions étant bien séparées les unes des autres, le texte présenté est assez lisible et très riche en informations. Cette opération est en général appelée ***désassemblage*** ; le nom de **décodage** lui conviendrait beaucoup mieux. Sous DOS, on obtient ce résultat par la commande **DEBUG** (cf. chap. VIII, p. 67).

8 - LE PREAMBULE

Le préambule est nécessaire à tout programme écrit en symbolique, non pas pour produire des instructions *exécutables*, mais pour fournir au *codeur* des informations indispensables à son travail. Les instructions y figurant s'appellent plutôt **directives** ou pseudo-instructions. On a déjà mentionné ci-dessus la nécessité de préciser à ce niveau le type d'objet représenté par les symboles ou variables.

Cette précaution est obligatoire dans le cas des **objets composites**, par exemple avec les *chaînes* de caractères ou les *tableaux*. Sinon, la machine serait incapable de leur réserver en mémoire la place nécessaire. Le programmeur pourra en même temps attribuer une *valeur initiale* aux objets ainsi définis (composites ou non), ce qui provoquera à l'étape du codage non seulement l'attribution de la place nécessaire en mémoire, mais également l'écriture en cet endroit des valeurs indiquées (on appelle cela l'**initialisation** des variables).

Parmi les autres directives du préambule, figurent celles sur l'implantation du programme en mémoire et en particulier sur sa segmentation. Le programmeur déclarera en particulier si les données doivent occuper un autre segment que celui du programme. Par contre, s'il peut – ou doit – donner au début de son programme une adresse relative au segment occupé, il est bien rare qu'il puisse fixer le numéro du segment affecté à son programme (seuls l'*éditeur de liens* et le *chargeur* sont capables de générer des adresses complètes ou *absolues*, c.-à-d. segment compris).

9 - LA GESTION DE LA PILE

La pile *(stack)* est une mémoire de taille modeste, faisant souvent partie de la mémoire principale. La taille de la zone réellement occupée dans la pile varie à tout instant. Mais elle est toujours connue de l'UAL grâce à deux *pointeurs*, celui de début de pile et celui de la dernière cellule occupée. On parle quelquefois d'adresses de haut et de bas de pile. Ces deux adresses sont mémorisées dans des registres spéciaux de l'UC (appelés BP et SP dans les PC, *base* et *stack pointers)*.

Cette mini-mémoire est très utile au programmeur, son emploi étant on ne peut plus simple. Elle fonctionne comme la pile d'assiettes au restaurant : le plongeur place au-dessus les dernières lavées et le garçon les prend en premier pour les besoins du service. En informatique, ce procédé s'appelle "LIFO" *(last in, first out*, ou *dernier arrivé, premier sorti)*. Ce n'est pas ainsi que devrait fonctionner la file d'attente chez le médecin, ni celle des dossiers sur le bureau de l'employé, où l'adage inverse, *premier arrivé, premier sorti*, devrait être la règle ! Des piles de ce type se rencontrent parfois en informatique sous le nom de FIFO *(first in, first out...)*. Mais revenons à notre pile LIFO.

Conçue ainsi, la pile se programme facilement. Veut-on sauvegarder les registres **AX**, **BX**, **DX** ... ou d'autres, pour les réutiliser plus tard ? On écrira :

PUSH AX PUSH BX PUSH DX .... (5)

Pour récupérer ces valeurs dans les mêmes registres, il suffira d'écrire, si on n'a rien empilé d'autre depuis :

POP DX POP BX POP AX .... (5)

On *dépile* les registres dans l'ordre inverse où on les a empilés. C'est un procédé très utilisé dans les programmes en LM (et l'un de ceux qui les rendent si puissants). Il n'est pas nécessaire que les registres de sauvegarde-récupération soient les mêmes. Seul l'ordre compte. Signalons au passage que la pile LIFO constitue par ailleurs la mémoire indispensable aux calculettes qui travaillent en *notation polonaise inverse* (6) (couramment utilisée par Hewlett-Packard).

On verra dans la section 11 que la pile joue également un grand rôle dans la transmission d'informations entre programme principal et sous-programmes.

10 - ADRESSAGE INDIRECT

Pour gagner en souplesse, les langages machine utilisent intensément le procédé d'adressage indirect. Il consiste à ne pas laisser à un unique registre le soin de composer l'adresse d'une donnée. Cette adresse est formée souvent par la combinaison de deux ou trois registres. On a déjà étudié le rôle de la segmentation (chap. IV, §1d-β). Des registres ordinaires peuvent ajouter des décalages supplémentaires dans le calcul de l'adresse définitive. Par exemple, l'adresse du *i*ème élément d'un tableau sera toujours obtenue par addition d'un registre contenant la valeur de *i* et du registre contenant l'adresse du *début* du tableau cité.

Une autre possibilité très courante consiste à placer dans un registre une valeur égale à l'adresse de la variable définitive. Ce registre sert alors d'*index*.

11 - APPELS DE SOUS-PROGRAMMES

Pour des raisons déjà développées, il est nécessaire de disposer de sous-programmes (SP). Appelés aussi *procédures*, ils sont dotés d'un nom et signalés comme tels au codeur par la présence de deux directives l'encadrant, telles que **PROC** et **ENDPROC**. Ils seront appelés dans le programme symbolique par le *mnémonique* **CALL** suivi de leur nom. Dans le programme codé, ils seront appelés par le code opératoire correspondant à **CALL** suivi de l'adresse-mémoire du SP (cette adresse étant en général fournie à l'étape *édition de liens*). On sait qu'une fois un SP exécuté (il comporte obligatoirement au moins une instruction de type **RET**, *retour*), l'exécution du programme appelant reprendra juste après l'instruction **CALL**.

Le sous-programme traitera les données transmises par le programme appelant ; il aura besoin pour cela des registres généraux. Mais le programme appelant doit retrouver, après le **CALL**, tous les registres dans l'état où il les avait laissés. C'est par la pile que le programmeur va résoudre ce problème, en lui confiant, au début d'un sous-programme, le contenu de chaque registre devant être utilisé. A la fin, juste avant le **RET**, le programmeur *dépilera* les registres sauvegardés. Ainsi, le programme appelant les retrouvera bien dans l'état où il les avait quittés. La même précaution est prise pour le registre d'état, mais cette sauvegarde est en général assurée par la machine elle-même. L'ensemble de ces mesures se nomme respectivement *sauvegarde* et *restitution du contexte*.

Ceci établi, programme appelant et sous-programme appelé *échangent* des données dans un sens ou dans l'autre, essentiellement par trois moyens :

(5) Bien sûr, à raison d'une seule instruction par ligne.

(6) Cette notation consiste à écrire (dans une pile), pour chaque opération, les opérandes **avant** l'opérateur. Ainsi, 31 ***a*** 4 + * ***sin*** signifie ***sin***[(4+***a***)*31] ; chaque opérateur extrait de la pile les éléments les plus hauts qui lui sont nécessaires, en fait ses opérandes, y place son résultat et se retire.

– par les **registres**, procédé le plus simple et le plus rapide. Avant l'appel, on place les données dans les registres appropriés. Le sous-programme les trouvera en place en début d'exécution. Le même mécanisme joue pour le retour. Son défaut réside dans la faible capacité des échanges ainsi permis.

– par des **variables** déclarées *globales*, c'est-à-dire *communes* à la fois au programme appelant et au programme appelé. C'est un procédé *a priori* simple, mais d'un intérêt très limité, en particulier parce que cette mise en commun est difficile à réaliser à des milliers de kilomètres de distance et à des années d'intervalle. Telle est pourtant bien souvent la situation des équipes ou des personnes respectivement créatrices et utilisatrices de sous-programmes.

– par la **pile**, qu'on va encore mettre à contribution, en y empilant avant l'appel les données, ou *paramètres*, que le sous-programme devra utiliser. Le nombre de données empilées est fixé par l'auteur du sous-programme. Ce dernier, après la nécessaire sauvegarde des registres, s'empressera de récupérer les paramètres transmis, non pas par des **POP** (qui dépileraient les registres qu'on vient de sauver !), mais par des **MOV** utilisant l'adresse précédente de la pile contenue dans le registre **BP** en lui ajoutant le décalage nécessaire.

Pour programmer en langage machine, il est indispensable de bien comprendre le mécanisme d'appel et retour des sous-programmes. La gestion de la pile est très délicate, avec la nécessité d'en allouer une zone déterminée à chacun des (sous-) programmes **(7)**. Nous ne détaillerons pas plus ces mécanismes, mais le candidat à la programmation en LM devra consacrer un effort important à l'étude de ces processus complexes dans des ouvrages spécialisés.

12 - ARCHITECTURES CISC ET RISC

Le jeu d'instructions dont dispose un calculateur est l'un des facteurs les plus importants de sa puissance (les autres éléments étant la fréquence de son horloge, la *largeur* de son bus, le temps d'accès de son disque dur...). Deux écoles s'affrontent en ce qui concerne la conception des unités centrales et leur jeu d'instructions.

Au début de l'informatique, le jeu des codes-machine disponibles était rudimentaire. Mais peu à peu, il s'est étoffé, allant jusqu'à traiter directement des chaînes entières de caractères **(8)** (faculté très utile pour écrire les logiciels de bureautique). Le nombre des codes dépasse nettement 100 pour les *processeurs* de la série 80*x*86, sans compter leurs nombreuses variantes. Ces codes deviennent forcément de plus en plus complexes et leur décodage de plus en plus long. Cette philosophie est appelée codage **CISC** *(complex instruction set computer)*.

Une autre école prône l'architecture **RISC** *(reduced instruction set computer)* dans laquelle le jeu de codes, réduit au minimum, atteint à peine 50. Ils peuvent être alors très courts et tous de même longueur. Il en faudra certes plusieurs pour accomplir le travail d'une seule instruction CISC. Mais leur simplicité et surtout leur homogénéité leur vaudra un décodage bien plus rapide. Le décodeur va traiter 4 à 5 instructions en même temps dans un processus en plusieurs phases appelé *pipe-line*. Cela permet à chaque instruction de ne "consommer" en définitive qu'une seule période d'horloge (20 ns par exemple) parce qu'elle aura été largement *préparée* dans le *pipe-line*. L'architecture de la puce RISC est différente de celle des puces CISC : on y dénombre au moins trois UAL, dont l'une réservée aux calculs d'adresses et une autre à ceux sur les nombres flottants **(9)**. De plus, pile et *antémémoire* (cf. chap. IV, §2c, p. 28) sont alors faites de registres rapides internes à la puce.

Ces atouts permettent un gain en vitesse de 2 à 3 par rapport aux UC relevant des mêmes techniques microélectroniques, mais agencés selon la méthode CISC. De telles architectures sont courantes sur les gros et moyens calculateurs *(stations de travail)*, mais il est possible qu'elles s'implantent bientôt dans le monde des calculateurs personnels.

Le lecteur ayant déjà programmé des **calculettes** aura noté une certaine similitude entre les codes-instructions employés dans ces machines et ceux exposés dans ce chapitre. C'est que, malgré sa diversité, l'informatique présente une grande cohérence dans ses lignes générales. Cependant, le fait d'avoir compris la philosophie du langage machine – objectif de ce chapitre – ne sera pas suffisant pour écrire des programmes opérationnels. Il faudra dépasser les idées générales et apprendre les spécifications des instructions et des directives du *processeur* en cause. On espère du moins que les notions de base développées ici, très souvent mal expliquées dans les ouvrages spécialisés, guideront le candidat programmeur dans cette étude difficile.

(7) Cette zone est comprise entre les adresses contenues dans BP et SP, registres qu'on doit réinitialiser au début et à la fin de chaque sous-programme.
(8) De telles instructions entrent dans la catégorie des instructions répétitives (§6a).

(9) Le calcul sur les flottants exige plusieurs opérations : normalisation, calcul sur les exposants, calcul sur les mantisses. Ces calculs peuvent être réalisés par l'UAL normale, avec de très nombreuses instructions élémentaires. Ils sont souvent réalisés par un coprocesseur spécifique et ce, beaucoup plus vite, certains même en parallèle. Dans le Pentium par exemple, ce calcul exige un seul cycle d'horloge.

EXERCICES

1 - Donnez les principales différences entre le langage machine numérique et le langage machine symbolique.

2 - Nombre, nom et rôle des champs en langage machine ?

3 - Citez quelques mnémoniques usuels et leur signification.

4 - Peut-on choisir librement le nom des étiquettes, des mnémoniques, des opérandes ?

5 - Quelles sont les exceptions au déroulement séquentiel du programme ?

6 - A propos du BCD, qu'appelle-t-on codes interdits ? Comment s'effectuent les opérations sur des BCD ? Pourquoi doivent-elles être effectuées sur 4 bits et non sur 8 ou 16 ?

7 - Rappelez à quoi sert la pile. Qu'est-ce qu'une pile LIFO ?

8 - Qu'appelle-t-on *sauvegarde du contexte*? Quelle est l'opération symétrique?

9 - Comment s'effectue la transmission des arguments entre programme appelant et programme appelé ?

10 - Trouvez la valeur prise par le registre AX après exécution de l'instruction suivante, très employée en LM :

```
XOR AX, AX
```

11 - Essayez d'écrire un programme machine calculant les racines d'une équation du second degré $ax^2 + bx + c$. Les données *a*, *b*, *c* sont des entiers contenus dans les doubles mots **da**, **db**, **dc** . Seule la partie entière de *x* nous intéressera (si ces nombres étaient des réels, on pourrait les multiplier par 10^3 ou 10^6, selon la précision recherchée).

Chapitre VIII

LES SYSTEMES D'EXPLOITATION

On pourrait utiliser un calculateur et ses périphériques avec le seul langage machine : on emploierait alors intensément des ordres d'entrée-sortie (**IN**, **OUT**) manipulant des octets souvent au niveau du bit. Tout utilisateur d'un ordinateur s'évite ce travail fastidieux en le munissant d'un **système d'exploitation** [(1)] ou, plus brièvement, un *système*, logiciel *ad hoc* écrit par une firme spécialisée. Une qualité primordiale du système d'exploitation (en abrégé SE ou OS en anglais pour *operating system)* est d'être le plus possible **indépendant** de la machine, de sorte qu'un changement de matériel ne se traduise pour l'utilisateur que par des modifications mineures s'il garde le même *système*. Aussi n'en existe-t-il qu'un très petit nombre.

Le **DOS** *(disk operating system)*, écrit par la société Microsoft pour les IBM-PC vers 1981, s'est taillé la part du lion dans le monde des SE pour calculateurs personnels, (en 1990, plus des trois quarts en sont dotés) [(2)]. Perfectionné presque d'année en année au cours de versions successives *(releases)*, il possède cependant quelques défauts conceptuels, dont nous parlerons dans la section 12 (page 68).

On reproche souvent au DOS son abord quelque peu rébarbatif. Aussi, des logiciels ont été élaborés pour servir d'intermédiaire (d'*interface*) entre lui et l'utilisateur. Ces logiciels utilisent des schémas, des images ou des *graphiques* beaucoup plus *parlants* et plus simples d'emploi qu'une liste de commandes indigeste. Nous dirons quelques mots de ces logiciels dans les sections 13 et 14 (p. 66-69).

Parmi eux, pourrait figurer Windows, logiciel conçu à l'origine pour atténuer l'aspect sévère du DOS et rivaliser avec le Macintosh. Cependant, petit à petit, Windows, gêné par les limitations du DOS, est devenu un système d'exploitation à part entière, beaucoup plus puissant que le DOS, mais capable de "gérer" le DOS lui-même ainsi que les logiciels *(applications)* conçus pour lui.

Parmi les **concurrents** du DOS, on citera le SE gérant les *Macintosh* (près de 10% des ordinateurs) et ceux rencontrés sur les stations de travail ou sur les calculateurs personnels haut de gamme : *OS2*, *VMS* et *Unix*. Nous en parlerons dans la section 16, mais le présent chapitre sera presque exclusivement consacré au DOS. Il suffit d'ailleurs à un informaticien d'avoir bien compris ce SE pour gagner un temps précieux dans l'étude de la plupart de ses concurrents.

1 - LES INTERRUPTIONS OU FONCTIONS SYSTEME

Le DOS repose sur un jeu de quelques dizaines de **fonctions** gérant mémoire centrale et périphériques. Ces fonctions, numérotées de n=0 à n=255, sont appelées (dans un programme en langage machine) par une instruction de type **INT *n*** *(interruption n)*. En toute rigueur, il y a là abus de langage, le nom d'*interruption* convenant plutôt à des événements extérieurs aléatoires capables de dérouter le programme. Nous en avons parlé dans le chapitre IV (pages 29-30). Les fonctions du DOS n'ont rien d'aléatoire, mais leur mise en œuvre étant tout à fait semblable à celle des vraies interruptions [(3)], elles en ont pris le nom.

Le processus de **déclenchement** des interruptions est simple : lorsqu'elle rencontre dans un programme l'instruction machine **INT** *n*, l'unité centrale va lire les deux mots de la mémoire vive dont l'adresse absolue est 4n et 4n+2, situés *tout en bas* de la mémoire, dans une zone appelée *table des vecteurs d'interruption*. Dans ces deux mots, la puce lit une nouvelle adresse, celle de la fonction-système numéro *n*. Elle exécute alors cette fonction après bien sûr sauvegarde du contexte. Ce procédé permet à un informaticien ou à un concepteur de logiciel d'écrire des fonctions personnelles, plus conformes à ses souhaits : il pourra les implanter en mémoire à une adresse quelconque, mais il devra modifier le contenu du vecteur d'interruption correspondant, c.-à-d. remplacer aux adresses 4n et 4n+2 l'adresse d'origine par celle de sa fonction personnelle (dans l'ordre, déplacement et segment). Autant dire tout de suite que de telles opérations ne sont pas conseillées aux informaticiens débutants.

---

(1) Cette appellation n'est vraiment pas significative. Peut-être le terme de *moniteur* serait-il plus approprié.

(2) Selon la revue *Micro-systèmes* (octobre 91, page 194), cela représentait à cette date un parc supérieur à 100 millions de machines.

(3) Les interruptions *vraies* (celles provenant des périphériques) font en réalité partie de l'ensemble beaucoup plus vaste des interruptions au sens large (fonctions DOS). L'interruption *vraie* de numéro m est assimilée à la fonction $n = m+8$. D'où l'adresse $4(m+8)$ de son vecteur, mentionnée dans la légende de la figure 4-4.

2 - LES COMMANDES SYSTEME

Pour l'utilisateur ordinaire, le recours de loin le plus naturel au système d'exploitation consiste à taper au clavier l'une de ses **commandes**. Celles du DOS sont réparties en deux groupes : les commandes *internes* ou *résidentes* qui font partie du fichier appelé **COMMAND.COM** et les commandes *externes*, contenues chacune dans un fichier portant le nom même de la commande. Ces commandes ou **ordres** sont autant de petits programmes qui appellent une ou quelques-unes des *fonctions* du DOS du paragraphe précédent.

La plupart des commandes acceptent ou exigent des arguments (noms de répertoires, de fichiers) et des paramètres (toujours facultatifs, précédés du symbole **/** , ils spécifient l'une des variantes de la commande). L'ensemble commande, arguments, paramètres s'appelle la **ligne de commande**. Elle se termine par RC (*retour chariot, Enter, Entrée...*).

En général les SE autorisent les utilisateurs à **créer leurs propres commandes**. Celles-ci doivent être écrites chacune dans un fichier portant le nom de la nouvelle commande ; elles seront exécutées ensuite à la simple frappe de ce nom. Deux restrictions toutefois : *primo*, le nom de ce fichier doit satisfaire un symbolisme spécial (terminé par les lettres **BAT** en DOS) ; *secundo*, ce fichier ne peut être constitué que d'une liste de lignes de commandes (d'origine ou personnelles). On peut par ce biais affecter à une commande du système un nouveau nom, plus significatif, en réduisant sa liste à cette seule commande. Dans tous les cas, c'est une facilité extrêmement utilisée. De tels **fichiers de commandes** sont souvent appelés fichiers *batch* ou *fichiers de traitement par lots* (*scripts* sous Unix).

La majeure partie des commandes du SE concerne la gestion des fichiers dans les mémoires de masse. Nous allons expliquer comment s'opère cette gestion sous DOS.

3 - VOLUMES, REPERTOIRES, FICHIERS

Sans méthode, il serait difficile de retrouver un fichier dans l'étendue des mémoires de masse. On appelle **volume** l'un des *supports physiques* de ces mémoires : disque dur, disquette, bande magnétique, disque optique ... sont autant de *volumes*. Dans chaque volume, on crée des **répertoires** *(directories)*, pouvant contenir des fichiers ou être subdivisés eux-mêmes en d'autres répertoires appelés **sous-répertoires** *(subdirectories)* et ainsi de suite... Les répertoires se classent donc sur différents niveaux. On obtient une structure hiérarchisée ressemblant à celle d'un arbre généalogique. La souche s'appelle *racine* (*root*) ou *répertoire principal*. Sous DOS, son symbole est la barre inverse (****).

Des fichiers peuvent être rattachés à n'importe lequel de ces (sous-) répertoires, comme l'indique mal la fig. 8-1, où, pour plus de clarté, les fichiers ne sont placés qu'en niveau 3. Les noms des fichiers et ceux des répertoires sont écrits ici respectivement en minuscules et en majuscules pour les distinguer ; mais il ne s'agit pas d'une convention du DOS, qui remettrait, quant à lui, tous ces noms en majuscules.

Il faut savoir comment le DOS range ses fichiers. Il divise les volumes en *unités d'allocation (clusters, UA* en abrégé), d'une taille de 0,5 à 8 kilo-octets selon le volume. Le DOS répartit les fichiers dans les UA libres, à mesure qu'il les rencontre. En effet, si on a modifié un fichier en augmentant sa taille, le DOS ne pourrait plus le reloger à l'emplacement précédent devenu insuffisant. Par ailleurs, la suppression d'un fichier libère de la place. La réutilisation des UA libérées et la *fragmentation* des fichiers contribuent à une gestion efficace des mémoires de masse **(4)**. Pour retrouver les fichiers *éparpillés*, le DOS crée une table (en deux exemplaires pour plus de sûreté), dite FAT *(file allocation table)*, qui décrit, UA par UA, toute l'affectation du volume. Cette table est située au début du disque, c'est elle que le DOS consulte quand on lui demande de retrouver un fichier.

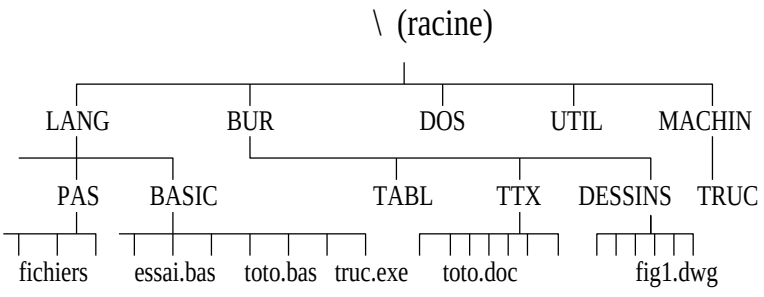


Fig. 8-1 : Structure d'un volume de mémoire de masse.

(4) L'UA étant la taille minimum pour un fichier, les tout petits fichiers gaspillent la mémoire sur disque et disquettes. Attention aux trop nombreux fichiers de commandes.

4 - NOMS DES UNITES, VOLUMES, REPERTOIRES, FICHIERS

Sous DOS, les **unités** de lecture-écriture sont désignées par l'une des premières lettres de l'alphabet, suivie du symbole "deux points". Ainsi, **A:** désigne le premier lecteur de disquette, **B:** le second s'il existe, **C:** le premier disque dur, les lettres suivantes représentant soit une autre unité de disque dur, soit une partition du premier, soit un *disque virtuel* (partie de mémoire vive déclarée comme devant être assimilée à un disque dur), soit une unité de cassette... La dernière lettre utilisée doit être communiquée au système (via le fichier **CONFIG.SYS**). Dans la suite de ce manuel, on désignera toute *unité* de ce type par la *lettre générale* **U:** . Il faudra se souvenir qu'une telle appellation doit être remplacée par **A:** , **B:** , **C:** , etc., et qu'elle désigne indifféremment l'*unité* de lecture-écriture ou bien le **volume inséré en elle.**

L'utilisateur donne à ses *répertoires* le nom qui lui plaît, avec, sous DOS, les restrictions suivantes : ce nom ne doit pas contenir plus de **8 caractères valides**, c.-à-d. choisis dans le jeu autorisé (lettres, chiffres, quelques symboles comme $, &, #,-, _ ...). Il n'y a pas de différence entre majuscules et minuscules.

Le **nom d'un fichier** comprend deux parties : la première (ou **partie principale**) obéit aux mêmes règles que le nom des répertoires. La seconde, ou **suffixe**, est facultative,[5] mais fortement recommandée . Séparée de la première par un point, elle comprend au maximum trois caractères parmi ceux autorisés. Ce suffixe (*extension*) est normalement employé pour indiquer le type du fichier, ce qu'il contient, ou bien le type de logiciel qui l'a créé. Par exemple, le suffixe **PAS** signale presque à coup sûr des fichiers-programmes écrits en Pascal, **BAS** en Basic, **FTN** ou **FOR** en Fortran, **C** en C..., **DES** ou **DWG** désigneront adroitement des programmes de dessin, **TXT** ou **DOC** des fichiers de texte... Certains suffixes sont réservés au système et ne doivent pas être utilisés au gré de l'opérateur, comme **EXE** et **COM** (qui désignent des fichiers exécutables en langage machine), ou **BAT** (fichiers de commandes), ainsi que **SYS**, **BAK**, **OBJ**, **BIN**, **CHK**, **LIB**, **LST**, **MAP**... déjà employés par des logiciels courants (compilateurs par exemple). De même, en ce qui concerne la partie principale du nom des fichiers, quelques appellations sont réservées, celles des unités d'entrée-sortie normalisées : **CON** (la console et l'écran), **PRN** ou **LPT** (les imprimantes), **AUX**, **COM** (portes de communication sérielle)...

Le **nom complet** d'un fichier comprend le nom **U:** de l'unité dans lequel son volume est inséré, puis le signe **** *(barre inverse, antislash)* et le nom de tous les répertoires de l'*arborescence* joignant ledit fichier à la racine, ces noms étant chacun suivis du signe **** . Cette suite est souvent appelée *chemin (path)*. Viennent ensuite le nom principal du fichier et, s'il en est muni, son suffixe précédé d'un point.

Ainsi, dans la fig. 8-1, le nom complet ou *absolu* du fichier **ESSAI** est **U:\LANG\BASIC\ESSAI.BAS .** Cette dénomination, qui est valide en toutes circonstances, apparaît cependant bien lourde. La notion de *répertoire actif* va simplifier la dénomination des fichiers et répertoires en introduisant une notation relative et non plus absolue.

5 - VOLUME ACTIF, REPERTOIRES ACTIF ET POTENTIEL

Les systèmes d'exploitation manipulent une sorte de *curseur virtuel* qu'ils déplacent d'un répertoire à l'autre. Le DOS gère **un curseur par unité** et ce, au moyen des deux commandes suivantes :

U: (avec **U** = **A**, **B**, **C**, **D**, ...),
CD *repert* (où *repert* est le nom d'un répertoire).

On appelle **unité active** l'unité désignée dans la dernière commande de type **U:** écrite au clavier ou lancée par un fichier **.BAT**. Cette commande désactive la précédente unité active.

On appellera dans ce livre **répertoire potentiel** de l'unité **U** celui figurant dans la dernière commande **CD** s'appliquant à cette unité, soit parce qu'elle y est explicitement désignée, soit parce qu'elle est active à ce moment-là.

On appelle **répertoire actif** (ou *courant*) le répertoire potentiel de l'unité active. La série de commandes suivante éclairera ces notions :

A:	désigne l'unité de disquettes **A** comme unité active.
CD A:\TOTO ou **CD TOTO** ou **CD \TOTO**	désigne le répertoire **TOTO** comme répertoire actif, à condition qu'il soit de niveau 1 (c.-à-d. rattaché à la racine \ de la disquette).
CD C:\PROGR	rend **PROGR** répertoire *potentiel* dans le disque dur **C** (mêmes réserves).
C:	rend le disque dur unité active. Le répertoire **PROGR** de **C** devient répertoire actif et **TOTO** répertoire potentiel dans **A .**

(5) Dans le Macintosh, seul le nom principal du fichier apparaît à l'écran. Une partie de l'information (celle du suffixe sous DOS), n'est visible que grâce à un *utilitaire*, qui fera apparaître le nom du logiciel ayant créé le fichier. Le *system* utilise amplement ces précieuses informations.

Le DOS autorise l'emploi d'appellations (*chemins*) **incomplètes** pour les fichiers et répertoires. Un nom est incomplet s'il ne commence ni par **U:** , ni par ****. Il est alors **relatif au répertoire actif** : le premier nom cité dans le chemin doit être celui d'un fichier ou d'un sous-répertoire rattaché directement à lui. On peut dire que le DOS ajoute devant ce nom incomplet le chemin du répertoire actif : le nouveau nom formé doit constituer l'appellation absolue d'un fichier existant.

Seule exception à cette règle : si le nom du fichier débute par **** , le DOS ne le fait précéder que du nom de l'unité active. Les ordres ci-dessus **CD TOTO** et **CD \TOTO** utilisent des chemins incomplets, mais, le répertoire **TOTO** étant situé dans le répertoire actif **A:**, la commande est valide et sera correctement interprétée.

On dispose là d'un premier moyen pour désigner des fichiers de manière raccourcie (ainsi que pour retrouver un répertoire sur une unité ayant été désactivée). Un second moyen sera donné par les symboles ci-après.

6 - APPELLATIONS SYMBOLIQUES

Le DOS offre quatre symboles pour faciliter l'écriture des chemins ou des fichiers. On a déjà mentionné le signe **** qui, s'il est placé en tête ou juste après le nom de l'unité, désigne le **répertoire racine**.

Le symbole **..** (deux points successifs) permet de désigner le *répertoire-père* du répertoire actif ou du répertoire désigné, i.e. de remonter d'un niveau dans l'arborescence.

Par exemple, pour rendre **TABL** actif depuis le répertoire **PAS** (se référer à la fig. 8-1, page 62), on pourra écrire :

CD \BUR\TABL *ou* **CD ..\..\BUR\TABL**
ou bien **CD ** *puis* **CD BUR\TABL**
ou encore **CD..** *puis* **CD..** *et* **CD BUR\TABL**

Les deux derniers symboles, * et **?** (appelés *jokers*, *symboles génériques*, *symboles polyvalents*...) peuvent figurer dans le nom d'un répertoire, dans le nom principal d'un fichier ou bien dans son suffixe. Le symbole **?** *remplace* n'importe quel **caractère** valide, mais **un seul**, ce caractère pouvant être inexistant. Ainsi le nom **ma?s.bas** signifie-t-il aussi bien **mas.bas** , **maqs.bas** , **mars.bas** , **ma2s.bas** , s'ils existent sur le chemin désigné ou sous le répertoire actif. De même, **ma?s.ba?** désigne aussi bien les précédents que **ma9s.bar**, **ma&s.ba$** ...

Le symbole * *remplace* quant à lui toute **chaîne** de caractères donnant avec ceux spécifiés *avant lui* un nom de fichier valide (* doit être le dernier caractère du nom ou de son suffixe). Ainsi, s'il termine un nom de 5 caractères, il peut remplacer zéro, un, deux ou trois caractères successifs valides. Par exemple : dans le répertoire spécifié ou, sinon, dans le répertoire actif,

z*.bat représente tout fichier dont le nom commence par **z** et possède le suffixe **bat** ,

abc*.* représente tout fichier dont le nom commence par **abc** , quel que soit son suffixe, tandis que

. représente, sans exception, tous les fichiers du répertoire spécifié ou, sinon, tous ceux du répertoire actif.

Dernière remarque, les symboles **?** et * ne peuvent pas être employés dans n'importe quelle *commande*, mais seulement dans certaines.

7 - LES REPERTOIRES PRIVILEGIES

Le DOS possède une commande – **path** – qui permet de désigner certains répertoires comme privilégiés. Ainsi :

```
path c:\lang\basic; c:\bur\TTX
```

déclare **privilégiés** les deux répertoires du disque dur **basic** et **TTX** . Il est recommandé de faire figurer une telle commande dans le fichier **AUTOEXEC.BAT**, dont le rôle sera précisé section 11 (page 67).

On apprendra (§ 8) que le système d'exploitation, lorsqu'il reçoit une commande, ne recherche le fichier correspondant que dans une liste de répertoires bien précise. La commande préalable **PATH** permet d'ajouter à cette liste les répertoires spécifiés. Ce procédé permet d'écrire directement le nom d'une commande sans mentionner le répertoire qui la contient (ou, plus exactement, qui contient le fichier la représentant), pourvu qu'elle fasse partie d'un répertoire privilégié (on comprendra mieux à la page suivante).

Signalons que la même possibilité existe pour des fichiers de *données* avec la commande préalable **APPEND**, qu'on place de préférence, elle aussi, dans l'**AUTOEXEC.BAT**. Elle est cependant beaucoup moins utile que la commande **PATH** .

8 - INTERPRETATION DES COMMANDES

Le système d'exploitation est un meneur de jeu qui attend la frappe d'une ligne de commande au clavier, puis lance l'exécution de ladite commande. Cette exécution peut entraîner la "passation temporaire du pouvoir" à un autre programme. Ce programme, une fois exécuté, *redonne la main* au SE qui reprend l'attente.

L'état d'*attente* se manifeste par affichage à l'écran d'un caractère spécial (le symbole > pour le DOS, sauf modification) ou bien par celui d'une chaîne. Ce texte s'appelle l'**invite du système** *(prompt)*. L'invite est toujours précédée de la lettre symbolisant l'unité active et souvent, du nom complet du répertoire actif. Cette dernière information – très précieuse – s'obtient grâce à la commande **prompt pg** (à placer de préférence dans le fichier **AUTOEXEC.BAT** , comme toute commande générale).

Lancer une commande consiste à frapper le nom de cette commande, suivi éventuellement des arguments et des paramètres nécessaires (noms de fichiers, de répertoires, ...) et enfin de *valider* cette **ligne de commande** par la frappe de la touche *retour-chariot* (**RC**, *Enter*, *Entrée*, ou ↵), qui déclenche le processus d'exécution.

Le système d'exploitation se saisit alors du texte frappé et cherche à l'*interpréter* :

– il recherche d'abord si cette commande est une commande interne (c.-à-d. intégrée dans le fichier **COMMAND.COM**, implanté en mémoire vive),

– si ce n'est pas le cas, il cherche sur le disque un fichier de même nom, sans suffixe ou suffixé par **COM**, **EXE** ou **BAT** dans cet ordre, situé dans le *répertoire actif*,

– à défaut, il recherche si ce nom est celui d'un fichier, avec les mêmes suffixes, faisant partie de l'un des *répertoires privilégiés* (dont celui contenant les *commandes externes* du DOS, qui *doit* avoir été désigné comme tel).

Si cette recherche n'aboutit pas, ou bien si paramètres ou arguments ne correspondent pas à ceux attendus par la commande, un message d'erreur apparaît ainsi que l'invite qui autorisera une nouvelle frappe. Sinon, la commande est lancée.

Autrement dit, pour qu'un nom constitue une commande valide, il faut :

– que ce soit celui d'une commande **interne** du DOS,

ou bien

– celui d'un fichier traduit en langage machine et possédant le suffixe **COM** ou **EXE**, ou bien celui d'un fichier de suffixe **BAT** comportant lui-même une liste de commandes,

– **ET** que ce fichier soit contenu soit dans le répertoire actif, soit dans l'un des répertoires privilégiés.

Si on frappe, en guise de commande, un nom de fichier exécutable, avec son suffixe **COM**, **EXE** ou **BAT** ou bien en précisant son répertoire, le DOS restreint sa recherche au fichier ou répertoire ainsi désigné.

Ainsi, avec un disque hiérarchisé selon la fig. 8-1 page 62 et dont le fichier **AUTOEXEC.BAT** contient la commande **PATH** du §7, la frappe de **TRUC** après l'invite DOS fait exécuter sans ambiguïté le programme **U:\LANG\BASIC\TRUC.EXE** quel que soit à ce moment-là le répertoire actif. En effet, ce fichier **TRUC.EXE**, est le seul de ce nom et de suffixe **EXE**, **COM** ou **BAT**, qui soit contenu dans un répertoire privilégié, le répertoire **LANG\BASIC**.

9 - COMMANDES POUR VOLUMES ET REPERTOIRES

Peu de commandes se rapportent aux volumes (ou unités). On a cité la commande **U:** qui rend l'unité **U** active. La commande **ASSIGN U=V** permet de dérouter sur l'unité **V** toutes les commandes, ou presque, adressées à l'unité **U** (très utile quand un programme utilise un lecteur non disponible, ou pour permuter les noms d'un lecteur 5"¼ et 3"½).

La commande **FORMAT U:** initialise le volume contenu dans l'unité **U** en l'effaçant entièrement au préalable s'il n'est pas vide (prendre garde à ne pas appliquer cette commande au disque dur !). **LABEL** permet de donner – ou de redonner – un nom à un volume, tandis que **VOL** affiche ce nom.

Parmi les commandes concernant les répertoires : **(6)**

CD *repert* rend potentiel ou actif le répertoire *repert*
MD *repert* crée un nouveau répertoire de nom *repert*
RD *repert* le détruit (possible seulement s'il est vide)

La commande la plus usitée pour les répertoires est

DIR ou **DIR** *repert* ou **DIR** *nom.suffixe*

DIR provoque l'affichage de la liste des fichiers et des sous-répertoires du répertoire *repert*, s'il est nommément désigné, sinon celle du répertoire actif. Lorsqu'on précise un nom de fichier après **DIR**, l'affichage ne concerne que ce seul fichier ; mais si le nom comporte les symboles *polyvalents* **?** ou *, tous

(6) Les noms de ces commandes sont formés des initiales des mots anglais **Change**, **Make**, **Remove** et **Directory**.

les fichiers répondant à ce symbolisme seront affichés. Des informations supplémentaires (dates de création, attributs ...), peuvent suivre le nom des fichiers selon les paramètres placés à la suite de la ligne de commande. Par exemple, le paramètre **/W** fait afficher le nom des fichiers à raison de 5 par ligne d'écran (et **/P** une page à la fois).

On trouvera dans les manuels accompagnant le DOS d'autres commandes utiles comme **CHKDSK** (donne des informations sur un volume), **DISKCOPY** (recopie intégralement une disquette sur une autre *de même type*), **RENDIR** (renomme un répertoire), **BACKUP** (sauvegarde) et la commande réciproque **RECOVER** (restitue les fichiers sauvés). Se rappeler que l'*attribut* d'archivage **A** peut être utilisé par **BACKUP** pour ne sauver que les fichiers modifiés ou créés depuis la dernière sauvegarde. En effet, le DOS garde en mémoire non seulement la date et l'heure de la dernière modification de chaque fichier, mais il arme en plus l'*attribut* A quand ledit fichier est modifié et le désarme lors d'une sauvegarde par **BACKUP**.

10 - QUELQUES COMMANDES RELATIVES AUX FICHIERS

Commençons par une commande dangereuse, mais souvent nécessaire, celle de destruction. La commande

DEL *fichier*

détruit *fichier* (un nom incomplet est considéré comme relatif au répertoire actif). Avec les symboles **?** et * , on peut détruire plusieurs fichiers. L'opération la plus dangereuse consiste à écrire **DEL** *.* qui, après confirmation, supprime tous les fichiers du répertoire actif [7].

Arrêtons-nous sur la très importante commande **COPY**.

Tout d'abord, **COPY** permet de **recopier** un ou plusieurs fichiers, d'un répertoire dans un autre, qui peut être le même. Elle nécessite un ou deux arguments : le fichier *source*, puis le fichier *cible*. Les symboles **?** et * sont acceptés. Les noms de fichiers incomplets sont complétés par le chemin du répertoire potentiel de l'unité désignée ou par celui du répertoire actif si aucun nom d'unité n'est donné. Par exemple :

```
C:
CD C:\MACHIN\TRUC
CD A:\TOTO
COPY A: *.*
```

recopie dans **C:\MACHIN\TRUC** (répertoire actif) tous les fichiers rattachés au répertoire **TOTO** de la disquette placée en **A:**, mais pas ceux des *sous-répertoires éventuels* de **TOTO** (il faudrait pour cela écrire **XCOPY** avec le paramètre **/S**). Ce genre de commande est très utile pour *installer* sur le disque un logiciel nouveau. Plus simple est l'exemple suivant :

COPY A:\TOTO\TOTO_1.DOC ESSAI.BAS

qui recopie le fichier **TOTO_1.DOC** (répertoire **TOTO**, disquette **A**) dans celui nommé **ESSAI.BAS** appartenant au répertoire actif, en le surchargeant (on dit souvent *en l'écrasant)* s'il existe déjà, sinon en le créant.

COPY permet également de **concaténer** plusieurs fichiers, c.-à-d. de les fusionner en un seul. Ainsi :

COPY *toto* **+** *fich*.SUF* *ficres*

fusionne le fichier *toto*, puis tous ceux dont le nom répond à *fich*.SUF* en un seul fichier appelé *ficres*. Bien sûr, l'appellation polyvalente *fich*.SUF* peut être remplacée par le nom d'un fichier unique.

On peut créer et remplir de texte un fichier avec la commande :

COPY CON *fic*

suivie du RC (*Enter*) habituel, puis d'un texte quelconque, avec d'autres RC à chaque fin de ligne. Tout le texte sera inscrit dans le fichier *fic*, jusqu'à ce qu'on frappe *Ctrl* Z (touche *Ctrl* enfoncée pendant qu'on frappe Z. C'est le symbole de **fin de fichier**). Les *éditeurs* ou les logiciels de traitement de texte nous procurent heureusement des moyens beaucoup plus pratiques pour remplir un fichier.

Bien d'autres commandes sont disponibles pour les fichiers, par exemple :

FIND **/n** **"** *chaîne* **"** *fichier* recherche les lignes où apparaît le texte *chaîne* dans *fichier* et les affiche avec leur numéro ;

TYPE *fichier*, affiche à l'écran [8] le contenu de *fichier* (avec le paramètre **>PRN**, c'est sur le papier de l'imprimante que ce fichier est écrit ; ce paramètre **>PRN** après **DIR** permet d'imprimer la liste d'un répertoire) ;

PRINT *fichier* permet également l'impression du *fichier* cité ;

REN *fichier1 fichier2* donne à *fichier1* le nouveau nom *fichier2* ;

TIME, **DATE** et **VER** donnent l'heure, la date, le numéro de la version du DOS ;

CLS efface l'écran ;

(7) En réalité, la commande **DEL** n'efface pas physiquement les fichiers, mais détruit dans la FAT le lien permettant au DOS de les retrouver. Des logiciels (cf. §14) ou une commande DOS sont capables de les récupérer.

(8) La commande **TYPE**, mal conçue, fait défiler rapidement tout le fichier à l'écran. On peut arrêter ce défilement "au vol" avec la touche PAUSE. Mieux vaut cependant utiliser la ligne de commande **MORE <** *fichier* **.**

DEBUG *fichier*, suivie de la sous-commande **u**, affiche en LM quasi symbolique les premières instructions de *fichier* (ceci n'a d'intérêt que pour les fichiers de type **OBJ**, **EXE** ou **COM**). Continuer par **u**. **DEBUG** est une commande très puissante, capable par la sous-commande **a** de coder un programme symbolique en langage numérique. On peut s'en servir pour placer en mémoire (ne pas donner de nom de *fichier*) ou dans un fichier (donner son nom) une suite quelconque de caractères *hexadécimaux* : la sous-commande **d** permet d'examiner d'abord le texte original (16 octets par ligne, on obtient en tête l'adresse du premier: *segment:adr*, *adr* étant son adresse relative). Modifier le *n*ième octet par: **e** *adr+n octet*, puis recopier la modification dans le fichier par la sous-commande **w**. C'est ainsi qu'on peut introduire le caractère *Esc* (**1B**) à un endroit quelconque dans un fichier. Dans tous les cas, **DEBUG** se quitte en tapant **q**.

11 - DEMARRAGE DU CALCULATEUR

A la mise sous tension, le registre-instruction du PC reçoit l'adresse du **programme d'amorçage** *(boot)*, situé en mémoire morte (ROM). L'exécution de ce programme provoque la vérification de la mémoire vive et de la configuration (décrite dans la mémoire CMOS) **(9)**. Le PC active alors l'unité **A:**, puis tente de *charger* le BIOS et le DOS en mémoire vive. Le BIOS (*basic input output system*) est un logiciel qui adapte un PC quelconque au DOS, qui, lui, est relativement normalisé.

Pour effectuer le chargement du BIOS, puis du DOS, le programme d'amorce examine si l'unité active **A** contient un volume. Si elle est vide, c'est au tour de l'unité **C** d'être activée (voir note 6, page 72).

Dès qu'un volume a été détecté dans l'unité active, le programme d'amorce y recherche deux fichiers et les charge en mémoire vive : celui du BIOS (**IBMBIO.COM** ou **IO.SYS**) puis le noyau du DOS (**IBMDOS.COM** ou **MSDOS.SYS**) tous deux de type caché **(10)** (i.e. n'apparaissant pas avec la commande **DIR**) et nécessairement situés au début du volume. Si ces fichiers ne sont pas contenus dans l'unité active, un message d'erreur apparaît, déclarant *le système* introuvable. Il faut alors ou bien remplacer la disquette par une autre *contenant le DOS* ou bien simplement ouvrir le verrou de l'unité **A** pour que ces fichiers soient recherchés sur le disque dur (s'ils y sont effectivement implantés). Une fois *chargé*, le *système* "prend la main" et effectue encore des vérifications.

IBMBIO et **IBMDOS** (ou équivalents) s'exécutent ; le SE recherche s'il existe dans la racine du volume actif un fichier nommé **CONFIG.SYS**. S'il le trouve, il l'exécute. Ce fichier sert à installer en mémoire des *programmes résidents* (gestion de l'écran, du clavier, des caractères nationaux, ...), à préciser la *configuration* de l'installation et à indiquer éventuellement où se trouve le fichier *interpréteur* de commandes (instruction **shell**). En l'absence de **shell**, le SE recherche ensuite dans la racine et installe en mémoire vive **COMMAND.COM** comme *interpréteur*. Il exécute enfin, s'il existe, **AUTOEXEC.BAT**, fichier de commandes DOS écrit par l'utilisateur pour effectuer une série d'initialisations facultatives (déclaration des répertoires privilégiés par **PATH** par exemple).

Le système d'exploitation est prêt à travailler. Il affiche l'*invite* indiquant qu'il attend une *commande* (la forme de cette invite est modifiable par la commande **PROMPT**, qu'on place généralement dans **AUTOEXEC.BAT**, cf. page 65). La suite des opérations, c.-à-d. l'interprétation des commandes par le DOS, a été expliquée dans la section 8 **(11)**.

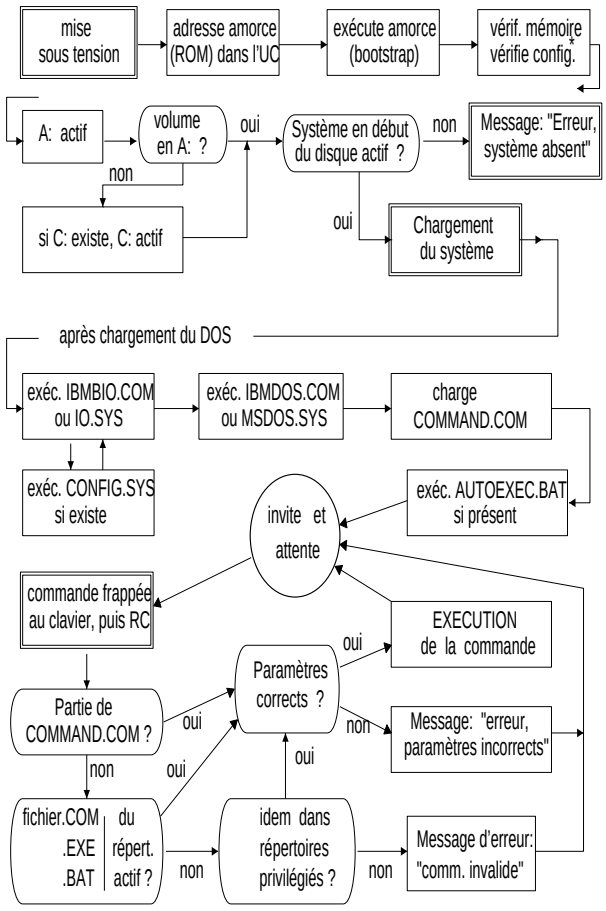


Fig. 8-2 : Chargement du DOS, initialisation du PC et interprétation des commandes.

(9) Un incident à ce stade (perte de la configuration) peut provenir de l'usure de sa pile. Le calculateur redemande alors la configuration : type des unités de disque, d'écran, taille-mémoire, présence d'un coprocesseur, date, heure ...

(10) Ces fichiers cachés interdits en écriture ne peuvent être copiés sur un volume *U* que par la commande **DISKCOPY** ou, lors de son *formatage*, grâce à la ligne de commande **FORMAT** *U*: **/s**, ou bien par la commande **SYS** *U*:

(11) Un redémarrage (avec réinitialisation partielle) peut être déclenché par appui simultané sur Ctrl-Alt-Del.

12 - DEFAUTS ET CONCURRENTS DU DOS

Le DOS dérive du système CP/M très répandu jusque vers 1980 – mais en perte de vitesse depuis – et il s'est perfectionné au cours de versions successives. Celles-ci ont l'énorme avantage d'assurer la **compatibilité descendante** : le DOS reconnaît en principe les fichiers et les logiciels écrits sous toute version antérieure (il y a parfois des problèmes, surtout avec les disquettes, dont la densité s'accroît au fil des ans et qui donc ne sont pas toujours exploitables sur un lecteur plus perfectionné que celui d'origine).

Cette compatibilité est très intéressante pour le client, qui n'est pas obligé – en principe! – d'acheter de nouveaux logiciels lors d'un changement de machine ou de version du DOS. Cependant, beaucoup de firmes proposent, pour leurs logiciels, lors de chaque nouvelle version DOS, des *mises à jour* (à prix réduit) qui sont mieux adaptées – et parfois indispensables – aux nouveaux moyens disponibles.

Cette compatibilité *historique* entraîne pour le DOS quelques graves inconvénients, car elle l'empêche de tirer parti de toutes les possibilités des nouveaux circuits de calcul (cf. note 2, chap. IV, p. 27) :

– il est inapte par principe à gérer plus d'un *mégaoctet* de mémoire (16 segments de 64 ko), dont seulement 640 ko sont accessibles au chargeur des programmes. Les palliatifs **(12)** proposés par les concepteurs de logiciels gourmands en mémoire ont l'inconvénient d'être peu normalisés et limités quant à leur intérêt,

– il est incapable d'exploiter les fonctions *multitâches* des nouvelles unités centrales (à partir du 80386),

– résolument mono-utilisateur, il n'assure pas un partage sans risques des ressources d'une installation. En particulier, il ne permet pas la protection des fichiers par des *mots de passe*.

Cependant, seul le premier point est vraiment gênant pour les utilisateurs du DOS sur un calculateur personnel.

Sur les installations plus "professionnelles" que les PC, le DOS n'est guère employé. Sur les PC haut de gamme, il le serait moins sans l'appoint de Windows, qui a suppléé aux défauts du DOS. IBM leur oppose un concurrent sérieux, OS2, système d'exploitation graphique (cf. §13), très convivial, perfectionné, compatible avec DOS et Windows. Il pourrait avantageusement les remplacer sur les PC, car OS2 fait exécuter les logiciels sous Windows plus vite que Windows. Peu de logiciels ont été écrits pour lui, mais sa compatibilité pallie ce handicap. On reparlera dans la section 16 de la rivalité entre Windows, OS2 et Unix pour la succession du DOS.

13 - LES SYSTEMES GRAPHIQUES

La seconde place parmi les systèmes d'exploitation est détenue par celui qui gère les Macintosh, dont il est indissociable. Ce SE relève d'une philosophie complètement différente de celle du DOS. Sans exiger la moindre commande, il se manifeste à l'écran par l'apparition de *fenêtres* qui sont autant de feuillets de travail se superposant ou se juxtaposant sur le *bureau* (l'écran). Ces fenêtres donnent la liste des fichiers (appelés *documents* chez Apple) contenus dans les répertoires *(dossiers)* qui obéissent à la même structure arborescente que celle décrite au sujet du DOS.

L'utilisateur sélectionne son *dossier* de travail, puis son *document*, en pointant la souris sur leur nom et en *cliquant*. Si des *options* doivent être précisées, une fenêtre de dialogue apparaît, dont la compréhension est la plupart du temps extrêmement facile. Si on demande à travailler sur un document déjà existant, le simple cliquage sur son nom provoque le chargement du logiciel *(application)* qui l'a créé, puis du fichier lui-même. Si, au contraire, on sélectionne une *application*, elle ouvre une fenêtre de sélection de fichier. Enfin, Apple a su imposer à tous les logiciels écrits pour la série des Macintosh le même type de dialogue, le même jeu de fenêtres et de *menus déroulants*. Les titres de ces menus sont affichés en permanence en haut de l'écran, tandis que leur contenu n'apparaît qu'après pointage-cliquage par la souris.

Ce système est caractérisé par l'utilisation intensive de dessins (formes graphiques). Des *icônes* viennent compléter la panoplie des moyens de dialogue, en représentant soit des fichiers (à un suffixe du DOS, il correspond maintenant une icône), soit des outils de travail. C'est parce que l'écran utilise le mode graphique et non le mode caractère qu'une telle richesse de dessins est rendue possible.

La manipulation d'un tel calculateur est extrêmement aisée et relève autant du jeu que de l'informatique. Cette philosophie est recommandée à qui ne veut pas s'impliquer dans cette discipline. Il aura à sa disposition un outil de travail parfaitement adapté à la *bureautique*, dont tous les logiciels, en sus de leur

(12) L'un de ces moyens – procédé LIM – consiste à décomposer la mémoire vive additionnelle en pages de 64 ko (*mémoire paginée*) qui sont chargées à tour de rôle par une fonction DOS dans une zone "haute" de la mémoire vive, zone accessible au DOS, mais située au-delà des 640 ko fatidiques (*cadre de pages*).

Un autre moyen, accessible à l'utilisateur, consiste à traiter la mémoire vive supplémentaire comme un disque, volatil certes, mais dont le temps d'accès est très court (*disque virtuel*). Avec ce procédé, les échanges sont moins rapides qu'avec le premier.

facilité d'emploi, offrent les services de la meilleure qualité qui soit. Les inconvénients consistent en un prix nettement plus élevé que ceux des compatibles IBM-PC, en la rareté des logiciels, la faible diversité du matériel, la pauvreté de la documentation sur le *système* (mais pas de celle sur les logiciels).... Difficiles à programmer, ces machines ne semblent pas avoir la souplesse des IBM-PC quand il s'agit de s'adapter à de nouveaux périphériques et encore moins s'il leur faut piloter de l'appareillage industriel.

La présentation par fenêtres graphiques étant d'un intérêt manifeste, les autres firmes créatrices de SE ont réagi en ce sens. Les systèmes les plus riches (OS2, Unix...) proposent maintenant tous cette philosophie. Le concepteur du DOS (Microsoft) est entré dans le jeu avec WINDOWS, qui au début se présentait plutôt comme une facilité annexe greffée sur le DOS. Jusqu'à sa version 3, WINDOWS était loin d'atteindre la perfection du couple *system-finder* du Macintosh. C'était cependant bien la voie à suivre pour les systèmes d'exploitation du futur et les dernières versions de Windows (gourmandes elles aussi en mémoire et en temps machine) sont très puissantes ; elles sont devenues des systèmes d'exploitation à part entière, mais compatibles avec le DOS. Entre-temps, des utilitaires semi-graphiques (interfaces) s'étaient immiscés dans le créneau laissé vacant. Ces logiciels gèrent les fonctions et les fichiers DOS de manière très agréable, mais moins perfectionnée qu'avec Macintosh. Bien que d'un intérêt moindre depuis les dernières versions de Windows, ces logiciels sont encore beaucoup utilisés. On va en parler un peu.

14 - LES AUXILIAIRES SEMI-GRAPHIQUES

Les logiciels présentés ici sont d'une puissance intermédiaire entre le DOS classique et WINDOWS. Il s'agit de logiciels ne prétendant pas remplacer le système d'exploitation, mais intervenant comme des auxiliaires présentant sous forme conviviale la plupart des possibilités du DOS. On appelle souvent ces logiciels des *interfaces* (mais ce terme est tellement galvaudé qu'il ne veut plus dire grand-chose).

Dans cette catégorie, les plus connus sont les logiciels *Norton Utilities* et *Pctools*. Ils présentent sous forme de cartes, tableaux ou fenêtres le contenu des volumes et des répertoires ainsi que les commandes accessibles (menus). L'affichage à l'écran n'utilise pas ici le mode graphique, mais seulement les caractères semi-graphiques du mode texte (cf. chap. VI §3, p. 44 et chap. V §3bγ p. 38), d'où beaucoup moins de finesse et de souplesse dans le graphisme.

Presque toutes les commandes DOS sont accessibles avec ces outils (quelques-unes cependant restent plus faciles à exécuter sous DOS). Ils utilisent la souris et savent lancer les fichiers exécutables avec sélection préalable d'un fichier de données qui sera ouvert par l'exécutable. Cet aspect les rend propres à superviser l'ensemble du travail sur l'ordinateur.

Ces logiciels permettent de plus d'utiliser des *fonctions* de base non accessibles par les commandes habituelles du DOS (cf. page 62). Il est possible par exemple de voir représentée en clair l'implantation réelle des fichiers sur un disque dur ou sur une disquette et voir alors leur fragmentation ; on peut faire apparaître le nom des fichiers cachés et les dévoiler en changeant leur attribut.

Une prestation spectaculaire de ces outils consiste en leur aptitude à **récupérer** des fichiers que l'on vient de détruire par une malencontreuse commande **DEL** (mais seulement avant que la place libérée n'ait été réaffectée). De plus, ils sont munis d'un *éditeur* (logiciel permettant de voir le contenu d'un fichier et de le modifier) beaucoup plus puissant que l'ancien **EDLIN** et plus rapide que l'actuel **EDIT** du DOS. Moyennant quelques additifs, comme la *concaténation* et l'*échange de blocs* entre fichiers, ils pourraient faire presque aussi bien qu'un système graphique, moins rapide et très gourmand en mémoire vive **(13)**.

15 - WINDOWS

Windows, à l'origine auxiliaire graphique de gestion du DOS, est maintenant un système d'exploitation puissant permettant d'exploiter les prestations des processeurs actuels. Il utilise les fonctions DOS, gère la mémoire étendue et autorise l'exécution de gros programmes et même de plusieurs à la fois. Ses menus et fenêtres accèdent aux fonctions DOS et assurent gestion des *documents* et lancement des programmes. On peut associer des suffixes de documents à des *applications* de façon à appeler cet exécutable par simple clic sur le nom du document (fichier produit par l'*application*).

Pendant l'exécution d'un programme, Windows permet d'inclure (importer) dans le document produit des références à un fichier créé par un autre logiciel et même de faire exécuter temporairement ce logiciel (pour modifier par exemple le fichier importé).

(13) Les versions DOS postérieures à DOS3 ont incorporé avec plus ou moins de succès un certain nombre des facilités offertes par les logiciels mentionnés dans cette section.

Windows travaille en mode protégé (cf. chap. IV, §2b), mais il peut faire exécuter tout logiciel écrit pour le DOS en mode réel. Il est multitâche et permet l'exécution simultanée de plusieurs programmes : l'un, privilégié, se voit affecter la fenêtre principale de l'écran pour ses sorties ; les autres, symbolisés à l'écran par leur icône s'exécutent en arrière-plan. Il comporte donc de sérieux avantages sur le DOS.

Ses détracteurs lui reprochent son étonnante lenteur et son occupation mémoire, surtout sur le disque dur, deux aspects défavorables par rapport aux auxiliaires semi-graphiques. Il est d'un emploi plus lourd et beaucoup moins intuitif que le *system* du Mac. Fortement "soutenu" par Microsoft et vendu à un prix modéré, il devrait cependant tôt ou tard faire disparaître le DOS, ou du moins sa partie visible.

16 - UNIX

Il y a peu, on assurait qu'Unix était le meilleur prétendant à la gestion correcte des PC de la nouvelle génération et qu'il assurerait la relève du DOS, devenu insuffisant. Aussi en dirons-nous quelques mots pour clore ce chapitre, bien qu'il soit encore plus rébarbatif (le nom de ses commandes est peu significatif) et nettement plus complexe que le DOS. Il est vrai qu'il possède de nombreux atouts sur lui, puisqu'il est *multitâche* et *multi-utilisateur*. Malgré tout, il manifeste avec lui une parenté certaine **(14)**.

Tout d'abord, la hiérarchie des répertoires et fichiers est semblable à celle du DOS, ainsi que la notion de répertoire actif. Les noms de fichiers peuvent comprendre 14 lettres (suffixe inclus, facultatif, mais peu employé). Les symboles * et **?** jouent le même rôle que sous DOS ; par contre la barre inverse de répertoire **** devient **/** . En particulier la racine de l'unité s'appelle *root* ou bien **/** .

L'utilisateur entre en relation avec Unix par la commande **login**, à quoi le système répond en demandant nom et *mot de passe*. Si les réponses sont correctes, le système *connecte* l'utilisateur sur le sommet de son arborescence personnelle.

Nombre de commandes se retrouvent sous une appellation différente, comme **ls** (pour **DIR**), **cat** (pour **TYPE**), **del** (**RM**), **cp** (**COPY**), **grep** (**FIND**), **pr** (**PRINT**), **mv** (**REN**)... D'autres ne changent pas de nom comme **cd**, **echo**, **mkdir** (**MKDIR** ou **MD** sous DOS) ...**(15)**

On retrouve la notion si utile de fichiers de commandes (avec la possibilité de *passer* des arguments : cf. exercices). Ces fichiers s'appellent des *scripts*. De même, au démarrage de la *session*, s'exécute automatiquement le fichier de commandes **.profile** (l'**AUTOEXEC** du DOS), mais il y en a maintenant un par utilisateur.

Le caractère *multi-utilisateur* se concrétise avec la notion de *super-utilisateur* (le chef de l'installation), des possibilités de communication entre les opérateurs (envoi de messages avec des priorités, à tous ou aux destinataires désignés ... grâce aux commandes **mail**, **mesg**, **write**, **wall**) et la notion de *protection*.

On peut, via **chmod**, autoriser l'accès d'un répertoire ou d'un fichier au seul *propriétaire*, ou bien à son groupe, ou à tous ; cet accès peut être limité à la lecture, à l'écriture ou même à la seule exécution.

Le caractère *multitâche* entraîne la notion de *processus* (travail) avec l'attribution d'un numéro *(pid)* à chacun d'eux par le *système*. Chaque processus, grâce à ce numéro, peut être orienté ou même annulé par des commandes spécifiques comme **kill**.

De nombreuses facilités supplémentaires sont offertes pour intervenir au sein d'un fichier (ce que le DOS assure très mal). De plus, deux *éditeurs* de texte sont mis à la disposition de l'utilisateur, l'un, **ed**, guère meilleur que **EDLIN**, l'autre, **vi**, éditeur plein écran, nettement plus intéressant, mais encore loin d'atteindre la convivialité des logiciels de traitement de texte modernes.

Précis et facile à programmer, Unix est bien implanté dans le monde des gros calculateurs et des stations de travail, où il règne en maître (il a supplanté VMS, SE déjà ancien produit par DEC). Mais il n'y a plus guère d'écart maintenant entre stations de travail et PC haut de gamme et Unix se trouve en compétition avec OS2, Windows et surtout Windows-NT. Unix possède des atouts et des handicaps. Son caractère ouvert (il ne dépend pas d'un constructeur unique) constitue un très gros avantage car il garantit la liberté de l'utilisateur ; par contre, ce même caractère lui vaut d'être peu normalisé et diffusé sous de nombreuses variantes. L'absence de normalisation est une préoccupation majeure de ses adeptes, qui se sont associés dans un organisme, l'OSF *(Open Software Foundation)*, pour tenter d'y remédier. La norme POSIX et son organisme de certification, le comité XOPEN, ont été élaborés, mais l'adhésion de la profession paraît loin d'être totale.

Les informaticiens amateurs attirés par Unix et son caractère professionnel peuvent s'en procurer gratuitement une variante, Linux, disponible sur certains serveurs publics.

---

(14) Il est antérieur au DOS et les concepteurs de ce dernier ont pu s'en inspirer.
(15) Attention : ces commandes sont seulement *proches* de celles du DOS et non identiques à elles.

EXERCICES

Les exercices qui suivent donnent des exemples d'application de quelques-unes des facilités du DOS.

1 - **Implantation des fichiers BAT en DOS**

Au lieu de disséminer les fichiers **.BAT** un peu partout dans l'arborescence, mieux vaut tous les grouper dans un même répertoire appelé par exemple **BAT**. Leur recherche s'en trouvera simplifiée. Les duplications seront plus faciles à détecter. De plus, si on a chargé les commandes DOS dans le répertoire DOS, la ligne **path** de l'**AUTOEXEC.BAT** se réduira à :

```
PATH C:\;C:\DOS;C:\BAT
```

2 - **Arguments des fichiers de commandes BAT**

Les *fichiers de commandes* (sous DOS, UNIX) acceptent des arguments tout comme les commandes du système. Dans ces fichiers, ces arguments s'écrivent **%1**, **%2**, ... Lorsqu'on lance la commande, on écrit le nom du ou des fichiers qui doit (doivent) se substituer à ces variables-arguments. Par exemple, supposons qu'on veuille créer la commande **lf** (lire fichier) affichant le texte d'un fichier à l'écran avec arrêt toutes les 25 lignes. Si l'on ne dispose pas d'un *éditeur* (**EDLIN** ou autre), on peut écrire au clavier la séquence ci-après :

```
CD \BAT
COPY CON LF.BAT
@ECHO OFF
MORE < %1
Ctrl Z
```

– La première ligne rend le répertoire **BAT** actif.

– La seconde dirige dans le fichier **LF.BAT** (qui sera créé s'il n'existe pas encore) tout ce qui sera frappé à la console (symbole **CON**), depuis le premier RC (retour-chariot) jusqu'au *Ctrl* **Z** final (par "*Ctrl* **Z**", il faut comprendre : frappe de **Z** pendant que la touche *Ctrl* est enfoncée).

– La 3e ligne stipule que le texte des commandes ultérieures ne sera pas affiché à l'écran au cours de leur exécution, l'arobasse (@) inhibant cette action pour la ligne en cours, c.-à-d. pour la commande **ECHO OFF** elle-même, qui sans cela, serait affichée, car elle ne prend effet qu'après son exécution.

– La 4e ligne signifie que l'argument (**%1**) de la commande **lf** sera transmis à la commande DOS **MORE** qui, ainsi écrite, atteindra le but visé, à savoir l'affichage du fichier écran par écran, à condition que l'on fasse suivre **lf** d'un nom de fichier valide, comme par exemple :

```
lf fichier.doc
```

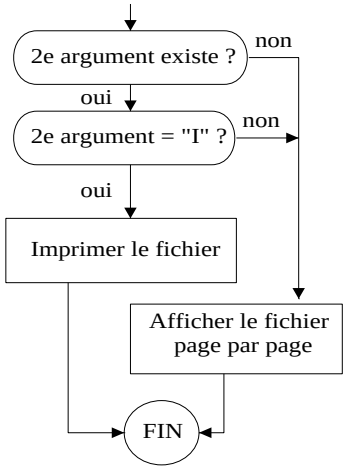
3 - **Fichier de commandes conditionnelles**

On pourrait "corser" le fichier **lf** précédent pour lui faire imprimer le *fichier* en question. Lors de la commande, on ajouterait simplement la lettre **i** facultative; on écrira alors :

lf *fichier* **i** (1)

Auparavant, il faudra remplacer, dans le fichier **LF.BAT**, la ligne **MORE %1** de l'exercice n°2 par les lignes suivantes :

```
IF NOT %2/==/ GOTO Ecran
IF NOT %2==i GOTO Ecran
TYPE %1 > PRN
GOTO Fin
: Ecran
MORE < %1
: Fin
```



L'organigramme ci-contre illustre la logique suivie. Les sélections sont un peu laborieuses à cause de la pauvreté du langage DOS. L'absence de second paramètre dans la commande (1) ne peut être sondée que par la première instruction **IF** ci-dessus.

A la suite d'un **IF** on ne peut placer qu'une seule instruction (comme en LM), ce qui oblige à passer par les **NOT** pour inverser les propositions. Remarquer la façon originale (et obligatoire) d'écrire les étiquettes **Ecran** et **Fin** .

A l'aide de fichiers de commandes acceptant des arguments variables, un informaticien suffisamment motivé peut recréer une sorte de système d'exploitation personnel qui se superposera à l'original. On parle alors d'une *couche logicielle* supplémentaire. Un tel procédé est surtout employé dans les grosses installations, leurs informaticiens réécrivant pour les utilisateurs nombreux et peu spécialisés des commandes plus conviviales que celles fournies par les constructeurs.

Pour chaque logiciel présent sur le disque, on pourra écrire un fichier **BAT** qui permettra de lancer le logiciel d'une manière simple et indépendante de l'emplacement du curseur des répertoires. En admettant que l'on dispose du tableur **ASEASY** dans le sous-répertoire **C:\BUR\TABL** et que les fichiers qu'il produit sont rangés dans le sous-répertoire **C:\BUR\TABL\DON**, on l'appellera par la commande **TAB** si on a auparavant écrit ainsi le fichier **TAB.BAT**

```
@ECHO OFF
CLS
C:
CD  C:\BUR\TABL
ASEASY  /h  /ATT  /DIR=C:\BUR\TAB\DON
```

liste reprenant les commandes qu'il aurait fallu frapper au clavier pour provoquer ce lancement. La commande **@ECHO OFF** empêche l'affichage à l'écran des commandes du fichier **BAT** lors de son appel. **CLS** efface l'écran. Derrière **ASEASY**, figurent les paramètres lui indiquant le type d'écran (ici ATT) et le répertoire des fichiers produits ou *documents* (**DIR =** ...).

Signalons également une ligne de commande très utile pour comparer le contenu d'un répertoire sur disque avec sa dernière sauvegarde sur disquette :

```
dir  /o:-d  /p
```

(elle pourra être sauvée judicieusement dans un fichier nommé **cat.bat**). Cette commande *décline* le répertoire (actif) page par page et **par ordre d'ancienneté**, les derniers fichiers modifiés étant placés en tête.

4 - **Redirections et filtres**

Les commandes DOS prévoient par défaut le clavier et l'écran comme organes d'entrée-sortie. Mais on peut en spécifier d'autres grâce à l'opérateur de sortie > (>> pour un ajout) et à l'opérateur d'entrée < . Ainsi :

DIR > PRN /w ou **DIR >>** *list.cat*

respectivement *décline* le répertoire actif (donne la liste de ses fichiers) sur l'imprimante (en condensé) ou l'ajoute dans le fichier *list.cat,* tandis que

MORE < *fichier*

affiche *fichier* page par page à l'écran. Il est possible également d'enchaîner des ordres avec des commandes-filtres *(pipes)* telles que **MORE** (affichage écran par écran), **SORT** (classe les chaînes d'un fichier par ordre alphabétique) ou **FIND** (affiche seulement les chaînes contenant la sous-chaîne spécifiée). Par exemple :

TYPE *fich* ¦ **FIND "***spécimen***"** ¦ **MORE > PRN**

sélectionne dans *fich* les chaînes où figure le texte *spécimen* et les imprime page par page.

5 - **Organigramme**

On n'a pas expliqué ce qu'est un **organigramme**. Il ne fait l'objet d'aucune norme. ; pourtant la plupart sont très compréhensibles. Les lignes fléchées indiquent le sens du déroulement du programme ; dans les rectangles, on inscrit les actions importantes, tandis que les figures ovales symbolisent des points de branchement. On y écrit la question ou la condition posée. Les lignes qui en émanent portent en exergue la teneur des réponses et indiquent les actions à entreprendre dans chaque cas. L'organigramme traduit en schéma ce qu'on appelle un **algorithme** de calcul ou de traitement, c.-à-d. le canevas des procédés logiques ou mathématiques élaborés pour parvenir au résultat désiré.

6 - **Note sur l'amorçage du DOS**

Dans les premiers PC, à la mise sous tension, le programme d'amorce recherchait impérativement *le système* d'abord dans l'unité A:. Avec les PC récents, on peut imposer (dans la mémoire de configuration ou *setup*) tout autre support comme unité de chargement du DOS.

Chapitre IX

LA PROGRAMMATION EN LANGAGE EVOLUE (première partie)

La programmation en langage machine est d'une telle lourdeur qu'on a bien vite senti le besoin d'en utiliser d'autres plus simples, c'est-à-dire plus proches des langues humaines. Ces langages, qualifiés de *haut niveau* ou d'*évolués*, permettent d'écrire des programmes formés d'instructions s'exécutant normalement *en séquence*, tout comme celles des langages machine. Toutefois, ces langages manipulent les *objets informatiques* de façon beaucoup plus globale, tout en adoptant une formulation très compréhensible.

Par exemple, l'impression de la ligne *texte*, pourra être provoquée par une instruction voisine de :

PRINT *texte*

alors qu'en LM, il aurait fallu créer une boucle dans laquelle on appellerait l'*interruption* appropriée après avoir chargé deux registres, l'un avec un numéro de fonction, l'autre avec les caractères successifs de la chaîne *texte*.

Ces langages comportent des *mots-clé* – ou *mots réservés* – et reconnaissent la plupart des opérateurs arithmétiques courants. Ils manipulent les *objets informatiques* qu'on a définis dans le chapitre VI (ou d'autres parfois) et acceptent que le programmeur désigne ces objets par des noms symboliques, souvent très expressifs (qui doivent cependant toujours commencer par une lettre).

Après un bref rappel historique sur la genèse de ces langages, on étudiera les grands concepts qui les rassemblent malgré leur diversité. On insistera davantage sur le plus populaire d'entre eux, d'ailleurs créé pour les débutants, le **Basic**.

1 - HISTORIQUE

Vers 1950, on s'aperçut vite qu'on aurait beaucoup de mal à commercialiser l'informatique, si on ne pouvait offrir à l'utilisateur que le langage machine (nous englobons dans ce terme les langages appelés *assembleurs*). Sans être tout à fait la première tentative en ce sens, le **Fortran** de John Backus *(formula translator)* fut le premier langage à permettre une communication aisée entre homme et calculateur. Dès 1955, grâce au soutien d'IBM, il conquit la majorité des scientifiques. Modeste et pragmatique à ses débuts, il évoluera par la suite en Fortran II (1957), Fortran IV (1962), Fortran 77, ... avec des améliorations à chaque étape. Cependant, il accuse son âge et s'est avéré mal adapté à l'essor de l'informatique personnelle. Il est encore beaucoup utilisé par les scientifiques, mais surtout pour des raisons de compatibilité avec les *bibliothèques* de programmes accumulés depuis plus de 30 ans.

Quelques autres langages ont dès le début essayé de concurrencer le Fortran, comme le PAF en France *(programmation automatique de formules)*. La série de machines CAB500 de la SEA (1958) en était d'ailleurs dotée.

L'empirisme du Fortran lui a valu de nombreux détracteurs, surtout en Europe ; ils se sont regroupés en 1958 pour définir un nouveau langage, beaucoup plus élégant et rationnel, l'**Algol** *(algorithmic language)*, qui n'a obtenu qu'un faible succès, n'étant pas soutenu par les grands constructeurs. Il sera cependant le père de toute une descendance, ironiquement appelée la *famille point-virgule* : le **Jovial**, imposé pendant longtemps par l'armée US, le **Pascal** de N. Wirth (1971), toujours très prisé surtout dans les Universités, puis le **C**, le **Modula-II** et l'**Ada**.

Défini en 1970 par Kernighan et Ritchie, le **C** est, parmi les langages évolués, celui qui se rapproche le plus du LM. Il est employé pour écrire des systèmes d'exploitation (dont Unix), les compilateurs des autres langages et la majorité des logiciels professionnels ou commerciaux. Gérant bien les ressources de la machine, il est en particulier très bien adapté à l'informatique personnelle ainsi qu'au *traitement* de l'information *en temps réel* (cf. chap. XV, §3, page 130).

Vers 1980, le *Department of defense* (USA) lança un appel d'offres pour la création d'un langage universel, modulaire, normalisé et apte à l'écriture de très

gros logiciels. C'est un groupe français de la société Bull, dirigé par J. Ichbiah, qui remporta le concours : son projet s'est appelé **Ada**, en l'honneur de Lady *Adélaïde* de Lovelace, la programmatrice de Babbage (cf. chap. I, note 5, p. 3). Héritier du Pascal, imposé par le DOF pour tous ses nouveaux logiciels – et, par voie de conséquence, par de très nombreuses firmes travaillant pour lui – l'Ada est promis à un bel avenir.

Les premiers langages étant orientés vers le travail scientifique, les administrations US éprouvèrent le besoin dès 1959 d'en créer un autre plus adapté aux problèmes de *gestion*. Ce sera le **Cobol** *(common business oriented language)*, encore très employé de nos jours, malgré la tentative d'IBM pour réunir gestion et science en un même langage universel, le **PL/1** (1965), qui n'a pas réussi à s'imposer (1).

Pour répondre à un autre besoin, on a développé ce qu'on appelle des **langages formels**, qui ne manipulent ni nombres ni texte, mais des **concepts**, structurés en *listes*. Dans cette optique, ont été créés le **Lisp** (1958), puis le **Prolog** *(programmation logique)*, langages bien adaptés à la description de certains problèmes de logique et en particulier de ceux relevant de l'**intelligence artificielle**. On peut classer dans cette lignée l'éphémère **Formac**, capable de calculs d'*algèbre formelle*, c.-à-d. littérale et non numérique (2), ainsi que le **Logo**, conçu pour l'initiation des écoliers.

Nous citerons enfin le cas particulier du **Basic** (***B**eginner **A**ll purpose **S**ymbolic **I**nstruction **C**ode)*, créé en 1965 par J.D. Kemenny et E. Kurtz dans la lignée du PAF. Conçu pour les débutants, il a été adopté d'emblée par les fabricants des premiers "ordinateurs de poche", puis par ceux des calculateurs personnels, dont les IBM-PC (3). Hélas peu normalisé, il diffère beaucoup d'une version à l'autre. Certaines d'entre elles lui confèrent une puissance digne des meilleurs langages, c'est le cas du Basic de Hewlett-Packard qui équipait bien avant 1980 les calculateurs de bureau de cette firme (séries 500, 9000, ...). C'est également le cas, mais dans une bien moindre mesure, de langages plus récents comme le *Quick-Basic* de Microsoft ou le *Turbo-Basic* de Borland.

2 - EDITION

Le premier geste du programmeur consistera à *appeler un éditeur* pour écrire son programme. Autrefois, il se serait assis devant un perforateur de ruban ou de cartes et alors, gare aux erreurs ! Il fallait dans ce cas recommencer à frapper toute la carte (80 caractères) ou tout le ruban, après recopie de la partie correcte du texte.

Actuellement, l'*éditeur* est un logiciel gérant le clavier et l'écran d'un ordinateur et permettant la *saisie* (l'écriture) d'un texte ne comportant que des caractères ASCII (le jeu réduit à 128 caractères ou bien ses *extensions 8 bits*). Il affiche les lignes *tapées* et permet de les corriger au droit du curseur-écran manœuvré à l'aide de la souris ou bien des touches flèches. Les éditeurs les plus simplistes (par exemple EDLIN du DOS) n'autorisent de correction que sur une seule ligne à la fois (c'est tout à fait insuffisant), les plus perfectionnés permettent de corriger n'importe quel caractère dans tout le programme. L'éditeur du GWBasic fourni avec le DOS avant sa version 5 est à mi-chemin entre ces deux extrêmes : seules une vingtaine de lignes à la fois sont accessibles à la correction ; on change ce sous-ensemble par la commande

list *n* – *m*

où les entiers *n* et *m* qui suivent le mot-clé **list** sont les numéros de la première et de la dernière ligne du sous-ensemble appelé à l'écran. En réalité, seules les 22 dernières lignes en seront visibles. Après correction, il ne faudra pas oublier de taper un RC *(retour-chariot, enter)* sur la ligne corrigée, condition nécessaire pour que la modification soit prise en compte. Pour qu'une ligne soit insérée dans un programme GWBasic, il lui faut **un numéro en tête**. Sinon, elle est considérée comme une commande, exécutée dès la frappe du RC.

Il faudra sauvegarder le texte du programme à la fin du travail et même de temps en temps au cours de celui-ci ; on le sauvegardera dans un fichier sur disque ou sur disquette. Ce fichier sera ensuite présenté (au minimum) au *traducteur*, grâce par exemple à une *ligne de commande* œuvrant sous le système d'exploitation. Comme on le verra bientôt, cette opération est bien plus facile si on travaille dans un *contexte de programmation intégrée*. Or l'*éditeur intégré* du GWBasic sur les PC n'étant pas très performant, il n'est pas rare de voir un programmeur en Basic utiliser un bon éditeur pour la première frappe ou pour les grosses modifications, alors qu'il procédera aux petites sous l'éditeur intégré (4). Avec la plupart des autres langages (y compris le QuickBasic), une telle jonglerie est inutile, car ils sont dotés d'un très bon éditeur.

(1) Dans les banques, il semblerait que l'on abandonne le Cobol au profit de l'Ada, parfois en passant par le Pascal.
(2) Dans ce langage, par exemple, pour l'opération **(x+a)*(x+a)** on obtient en réponse directement **x^2+a^2+2ax**, sans qu'aucune valeur numérique ait été affectée ni à *a* ni à *x*. Ce genre de langage, plutôt ardu, a cédé le pas devant des logiciels plus conviviaux, comme *Mathematica*.
(3) Ceux fonctionnant sous DOS sont dotés en série, avant le DOS-5, du GWbasic de Microsoft (ou du Basica d'IBM, tout à fait équivalent) et, ensuite, du QuickBasic.
(4) On quitte l'éditeur Basic du DOS en tapant **system** .

3 - TRADUCTION

Tout programme écrit en langage évolué doit être traduit pour être *exécuté*. Cette traduction est assurée par un logiciel qui transforme en LM le programme d'origine, contenu dans un *fichier source*. Le résultat de cette traduction, s'il est conservé, est placé dans un *fichier objet*. On distingue deux types de traducteurs :

– les **interpréteurs**, qui travaillent un peu comme les interprètes en *traduction simultanée* : ils traduisent le fichier source ligne par ligne, au fur et à mesure de sa lecture. Chaque ligne **est exécutée dès sa traduction** et le programmeur peut ainsi suivre assez facilement son déroulement pas à pas, ce qui facilite considérablement sa mise au point (cf. § 11, chap. X, p. 92).

– les **compilateurs**, qui traduisent **en bloc** le fichier source (la fig. 7-1, page 57, pourrait très bien illustrer ce processus, en modifiant ses légendes). Cette façon de faire entraîne des exécutions beaucoup plus rapides, pour de nombreuses raisons : par exemple si le programme comprend une boucle répétant 100 fois un bloc de 10 instructions, le compilateur ne le traduira qu'une seule fois, alors que l'interpréteur le traduira 100 fois (1000 instructions au total au lieu de 10).

La majorité des langages ne peut être que compilée. Le Basic, lui, est généralement interprété, mais dans ses versions les plus évoluées, il peut *également* être compilé.

Ces traducteurs vérifient d'abord le respect des règles du langage par les instructions du fichier source. Ils détectent les manquements à la **syntaxe**. Il ne faudrait cependant pas croire que la mise au point d'un programme est terminée quand il a franchi avec succès l'épreuve de la traduction : les **erreurs de logique ne sont jamais détectées**.

Les premières **règles de syntaxe** concernent la **présentation** des instructions. Les langages anciens (Fortran, Cobol), ont des règles très rigides à ce sujet et ne tolèrent pas par exemple plus d'une instruction par ligne d'écran (80 caractères, image des anciennes cartes d'IBM), ni que ces instructions commencent avant la colonne 7 de chaque ligne.

Les langages de la *famille Algol* exigent un **point-virgule** à la fin de chaque instruction et, à cause de cela, admettent une écriture très libre (ils acceptent plusieurs instructions par ligne). Une instruction peut occuper **plusieurs lignes**, ce qui ne soulève aucune difficulté dans la *famille Algol*, mais exige des marques spéciales de *continuation de ligne* en Fortran et en Cobol. Quant au Basic, il reconnaît comme symbole de fin d'instruction le caractère deux-points (:) et la fin de ligne (RC) ; on ne peut pas dépasser la longueur d'une ligne, mais elle admet 255 caractères !

4 - LES OBJETS MANIPULES

Un langage évolué peut manipuler des objets autres que ceux prévus par le langage machine. Mais il sera d'autant plus rapide qu'il emploiera ceux-ci en priorité. Bien sûr dans ce cas, le langage paraîtrait lié aux machines, ce qui lui serait imputé comme une tare. La solution de cet épineux problème réside dans la normalisation des objets informatiques, tels ceux décrits dans le chapitre VI. Il semble que ce soit le langage C qui se conforme le mieux aux objets normalisés ; c'est l'une des raisons de sa puissance. Le GWBasic, qui s'éloigne quant à lui beaucoup de ces objets, est peu rapide.

Le Fortran n'a pendant longtemps traité que des nombres, entiers ou réels [5]. Il reste mal adapté au maniement des bits, chaînes et structures. Le Cobol, lui, manipule bien les chaînes et les structures, mais quelques types seulement de nombres (quadruple mot de type entier, BCD de 10 octets). Le Basic "bas de gamme" lui ressemble, de ce point de vue, car il ne connaît que le quadruple mot (flottant en double précision ou DP) et la chaîne de caractères. Cette limitation de certains *Basic* à deux types d'objets contribue grandement à leur simplicité, mais elle se paye par un gaspillage de mémoire et un ralentissement des calculs (le calcul sur les flottants et les BCD est bien plus long que celui sur les entiers).

Chaque objet manipulé devra être appelé par **un nom**, qui joue le même rôle que celui des *variables* en algèbre. Ce nom doit respecter certaines **règles** et en particulier **ne jamais commencer par un chiffre**.

On a déjà dit qu'un **tableau** possède également un nom et un seul pour tous ses éléments, qui ne se distinguent les uns des autres que par un **indice** : ainsi **tab(22)** serait en Basic le 23e élément du tableau *tab*, le premier ayant l'indice zéro.

Certains langages [6] admettent les **structures**, collections d'objets disparates, chacun d'eux pouvant être lui-même un objet composite, tableau, chaîne, voire structure. Dans ce dernier cas, la (sous-) structure peut être de même type que la première : on entrevoit alors la complexité de tels objets, avec leur structure **arborescente** partant de *nœuds* et ramifiée

(5) Et même *imaginaires* ou *complexes*, au sens algébrique.

(6) On pourra éviter ce paragraphe en première lecture. Notez que les *structures* sont appelées *enregistrements* ou *records* en Pascal.

presque à l'infini. Ils sont très employés par les langages relevant de l'intelligence artificielle. Nous avons déjà rencontré ce type d'objet dans le chapitre précédent : les répertoires des systèmes d'exploitation sont des structures formées d'autres répertoires et de fichiers. On a vu qu'un élément de volume se désignait en citant les noms de tous les *nœuds* par lesquels il faut passer pour l'atteindre à partir de la racine. Il en va de même pour la désignation des composants de toute structure.

5 - DECLARATION DES OBJETS UTILISES

Dans tout programme relevant de la *famille point-virgule* ou *Algol*, il faut d'abord **déclarer** à quel **type** d'objet se réfère le nom de **toutes** les variables utilisées (ou **identificateurs**). Avec d'autres langages, cette déclaration est en partie facultative. Exemples :

char *a, numero, b3* **;**	déclare en C des variables-*octets*.
a, num, b4 **: float ;**	" en Pascal des *réels* (simples)
INTEGER *I0, J3, K*	déclare en Fortran trois *entiers*(7)
PI **PIC S9V9(4) COMP**	déclare en Cobol **PI** variable *signée* (**S**) en virgule fixe (**V**) avec 1 chiffre (1 symbole **9**) avant la virgule et 4 après.

Si le Basic n'exige pas la *déclaration* des variables simples *avant* leur emploi, c'est que les types d'objets autorisés sont identifiables par la présence d'un caractère spécial placé à la fin de leur nom : ainsi le **$** doit terminer les noms de chaînes, le **%** ceux des entiers et le # ceux des réels en DP. Sinon, l'objet est un **réel** simple. Le Fortran obéit à d'autres conventions : à moins de déclaration *explicite*, une variable dont le nom débute par *i, j, k, l, m,* ou *n* est un entier simple, toutes les autres sont des réels en simple précision.

En ce qui concerne les tableaux, la déclaration de leur type obéit aux mêmes règles que celles concernant les objets élémentaires dont ils sont composés. Par contre, on doit nécessairement (à une exception près) déclarer leur taille par des instructions telles que celles qui suivent.

int *total***[10][100] ; char** *toto* **[20][81] ;**	C
01 *TOTAL* **.** **02** *SSTOT* **OCCURS 10** **03** *ELEM* **OCCURS 100 PIC S9(5).** **01** *TOTO* **.** **02** *TOTO* **OCCURS 20 PIC X(80).**	Cobol
total **: array[1..10, 0..99] of integer;** *toto* **: array[0..19, 1..80] of char;**	Pascal "
20 DIM *TOTAL***% (10,100)** **30 DIM** *TOTO***$ (20)**	Basic(8)

Ces instructions déclarent, dans les langages énoncés à leur droite, que l'objet **total** est un tableau de 10×100 *entiers* (10 lignes de 100 colonnes, on dit que c'est un tableau à deux dimensions) et que l'objet **toto** est un tableau constitué de 20 *chaînes* pouvant contenir chacune 80 *caractères*. On remarquera la concision du C et du Basic (dans certains Basic, il n'est même pas nécessaire de déclarer les tableaux de moins de 11 éléments, c'est l'exception unique mentionnée à la page précédente). La déclaration d'un tableau en Cobol est particulièrement verbeuse, mais elle reflète une grande richesse, qui permet de désigner des sous-tableaux (y compris l'objet élémentaire lui-même) avec des noms différents de celui du tableau général ; en outre, dans ce langage, les mots-clés **PIC** ou **PICTURE** définissent le contenu et la présentation (le *format*) des variables.

Dans la plupart des langages, les objets peuvent recevoir des noms plutôt longs et donc suggestifs. Ce nom peut comporter 6 caractères en Fortran et plus dans les autres langages (8 à 40). En contraste, certains *Basic* rudimentaires n'acceptent que des mots formés de deux ou même d'une seule lettre : ils ne peuvent traiter qu'un nombre réduit de variables (26 dans le cas d'une seule lettre, un bon millier avec deux lettres, les chiffres étant admis en deuxième position).

La déclaration d'une structure est bien plus délicate que celle d'un simple tableau. La structure doit être décrite avec précision. Donnons en exemple la déclaration d'un tableau de structures qui, en C, permettrait de garder en mémoire les références de 366 jours calendaires :

struct { char *nom***[8] ; int** *num* **; char** *mois***[8] ; int** *an* **}**
jour **[366] ;**

Les 366 structures ainsi définies appelées **jour** regroupent chacune une chaîne de 8 caractères pour le nom du jour, un entier pour son numéro, une autre chaîne de longueur 8 pour le mois et un entier pour l'année.

En résumé, il faut bien garder à l'esprit la règle suivante : le compilateur (ou l'interpréteur) doit pouvoir déterminer exactement à quel type d'objet appartient chacune des variables utilisées, ainsi que sa

(7) Déclaration inutile en Fortran, puisque les initiales de ces variables les déclarent *implicitement* comme des entiers.

(8) En Basic, la déclaration d'un tableau de chaînes est délicate ; ici, les 20 chaînes seraient de type élastique (leur longueur limite dépend du langage, cf. pages 85 et 86).

taille, afin de lui réserver la place mémoire nécessaire. On rappelle que, pour cela, avec certains langages (C, Pascal...) toutes les variables doivent être déclarées en début de programme. Dans d'autres, des conventions d'écriture ou bien des conventions globales portant sur la première lettre de leur nom (Fortran, Basic [9]), ou sur la dernière (Basic), permettent d'affecter un type aux variables non explicitement déclarées, mais nombre de puristes considèrent ce procédé comme manquant de rigueur.

6 - AFFECTATIONS ET OPERATIONS ARITHMETIQUES

Le premier souci du programmeur consistera à donner une valeur à ses variables (alors que c'est tout à fait inutile en algèbre). Sans cela, celles-ci ne seraient pas manipulables ou pas *définies* (sauf en Basic, où elles auraient la valeur zéro). Voici des exemples d'*affectation* de valeurs :

j **= 99 ;**	*b3* **=** *j* **;**	(C)
j **:= 100 ;**	*b3* **:=** *j* **;**	(Pascal)
ALPHA **= 1515**		(Fortran)
B3 **=** *ALPHA*		(id)
55 *Gamma* **= 100 :**	*tintin* **=** *k*	(Basic)

Il faut bien comprendre le mécanisme caché derrière le symbolisme de l'affectation *(assignment)*. Ainsi :

$$a = b$$

veut dire : *allez chercher en mémoire la valeur de la variable b et placez-la dans a.* On ne sera donc pas surpris par l'écriture suivante, très employée en informatique (bien qu'interdite en algèbre) :

$$a = a + 5$$

qui signifie : *allez chercher la valeur de a , ajoutez-lui 5 et replacez cette somme dans a*. La séquence suivante (en Basic) :

15	*Toto* **= 80 :**	*Toto* **=** *Toto* **+ 12**
18	*B* **=** *Toto* **:**	*Toto* **= 100**

permet de donner la valeur 80, puis 92 à la variable *Toto*, puis 92 à *B*. Ensuite, *Toto* prendra la valeur 100, mais *B* gardera la valeur 92, les instructions s'exécutant, sauf avis contraire, dans l'ordre où elles sont écrites (15 et 18 sont des étiquettes).

Les affectations entre variables de **types différents** sont toujours délicates : selon le langage, elles peuvent être ou interdites, ou restreintes à des cas simples, ou bien totalement permises (comme en C). Le programmeur devra toujours connaître les règles propres au langage utilisé. Par exemple, l'affectation d'une valeur flottante (réelle) à une variable entière entraîne son **arrondi** en Basic, sa **troncature** en C, mais est **interdite** en Pascal (il faut utiliser une fonction).

Presque tous les langages reconnaissent les **symboles** des quatre **opérations** de base de l'arithmétique: +, –, *, / . Addition et soustraction ne soulèvent aucun problème. La **multiplication** est presque toujours caractérisée par le symbole * . La **division** entre nombres réels est conforme aux règles habituelles, mais celle entre entiers est source de nombreuses surprises. Sans précautions, on obtient en général la partie entière du quotient des deux opérandes. Les opérateurs **MUL**, **DIV** et **** sont parfois employés pour la multiplication et la division (en Cobol par exemple) ou pour caractériser certaines de ces opérations (par exemple, celles sur des opérandes de type entier en Pascal, Basic ...).

Tous les langages reconnaissent les **parenthèses** qui groupent des opérations à réaliser en priorité. En l'absence de parenthèses ou à l'intérieur de celles-ci, chaque langage utilise une table de priorités pour déterminer l'ordre d'exécution des opérations. Ainsi :

$$d = a * b + 15$$

est une expression ambiguë. Sa valeur est différente selon qu'on l'interprète comme **(***a*∗*b***)+15** ou comme *a*∗**(***b***+15)**. C'est généralement la première forme qui sera adoptée par les compilateurs (ou les interpréteurs), mais il vaut mieux utiliser les parenthèses, à moins de bien connaître la table des priorités du langage employé. Noter que dans les Basic rudimentaires, avec leurs 26 variables à lettre unique, l'écriture *ab* signifie *a*∗*b* . Hormis ce cas, à vrai dire peu courant, **le symbole de la multiplication est toujours obligatoire**.

Dans la plupart des langages, on dispose de l'opérateur **modulo**, qui donne le reste de la division (entière) en principe entre deux entiers. Il est souvent noté **MOD**, mais **REMAINDER** en Cobol (il suivra alors un opérateur **DIVIDE**) et **%** en C . Par exemple, si les nombres **11** et **3** sont des valeurs de variables entières, on obtiendra les résultats ci-après :

11/3 $\rightarrow$ **3** **11%3** ou **11 MOD 3** $\rightarrow$ **2**

Par contre, si, en GWbasic, on écrit l'instruction suivante

$$y = 25.7 \text{ MOD } 6.9$$

qui est parfaitement autorisée malgré la présence de réels, on obtiendra pour *y* la valeur 5, car les opérandes auront été arrondis respectivement à 26 et à 7, la division entière donnera 3 (3×7 = 21) avec pour reste 5.

(9) En Basic, par exemple **DEFINT i, j** implique que seront entières toutes les variables dont le nom débute par *i* ou *j*.

On retiendra que les langages informatiques se comportent **parfois** de manière bien différente entre eux. Dans le cas d'opérations délicates, comme celles relatives aux *divisions entre entiers*, il sera nécessaire de bien connaître les règles opératoires.

L'opérateur d'**exponentiation** est reconnu par presque tous les langages, mais singulièrement pas par le C, ni par le Pascal (qui, à sa place, utilisent une fonction). L'opérateur d'exponentiation se note soit ** (en Fortran, Cobol), soit ∧ (en Basic). Ainsi, *a**b* et *a*∧*b* signifient a^b.

On peut rencontrer dans les langages les plus évolués des opérateurs moins triviaux que les précédents comme, par exemple en C, des *opérateurs jumelés* qui réalisent deux opérations à la fois (et ne facilitent pas la lecture des programmes). Nous nous contenterons de mentionner leur existence et laisserons à l'apprenti programmeur le plaisir de les découvrir lui-même.

7 - OPERATIONS LOGIQUES ET BINAIRES

Les opérateurs logiques sont présents dans tous les langages. On citera tout d'abord les **opérateurs de relation**, tels que > , qui servent à établir des *comparaisons*. Leur signification est la même qu'en algèbre, mais ils s'écrivent parfois différemment.

Algèbre	=	>	<	≥	≤	≠
Basic, Pascal	=	>	<	>=	<=	<>
C	==	>	<	>=	<=	!=
Fortran (10)	.EQ.	.GT.	.LT.	.GE.	.LE.	.NE.

Les deux variables ou valeurs entourant l'un de ces symboles forment avec lui une **expression logique**, dont la valeur est *vraie* ou *fausse*. Ainsi, si les variables **delta** et **DELTA** sont strictement positives, les deux expressions de la ligne suivante sont *vraies*, tandis que les trois de la seconde ligne sont *fausses* (11) :

(*delta* **>= 0)** **(***DELTA***.GT.0)**

(*delta***<0)** **(***delta***==0)** **(–24.GT.***DELTA***)**

Grâce aux opérateurs logiques **AND**, **OR**, **NOT**, **XOR**, (cf. chap. II, page 10), on peut combiner ces expressions pour obtenir d'autres expressions logiques (qu'on peut appeler *composites*).

On appelle **opérations binaires** des opérations entre bits de même rang appartenant à deux objets différents. Les opérateurs les provoquant ont été étudiés dans le chapitre II (§5). Ainsi, en notation hexadécimale et en C :

A&3 est égal à 2, **A¦3** est égal à **B**.

puisque ces opérations – bit à bit – se détaillent ainsi :

```
 1010          1010
&0011         ¦0011
------        ------
 0010          1011
```

Ce type d'opération est très employé, spécialement pour mettre en forme les signaux chargés de piloter les périphériques. Elles portent souvent les noms de *filtrage, masquage, forçage, cryptage, décryptage ...*

Les opérateurs logiques ci-dessus (AND, OR ...) qui assurent des relations entre expressions logiques sont presque toujours capables d'exécuter également les opérations binaires (bit à bit) que nous venons de citer, du moins sur les objets (variables ou constantes) de types entier et caractère.

(10) Les symboles du Fortran sont les abréviations de *equal, greater than, less than, greater or equal, less or equal, non equal* . Remarquez que tous les langages, sauf l'algèbre et le Basic, marquent, par un moyen ou par un autre, la profonde différence entre une *affectation* et une *comparaison à l'égalité*.

(11) Ces expressions **logiques** (appelées aussi *booléennes)* sont utilisées surtout après **IF** ou dans les boucles de type **WHILE** (p. 88). Mais certains langages acceptent leur présence dans des opérations **arithmétiques** en leur donnant la valeur 0 si elles sont fausses et une valeur non nulle si elles sont vraies (par ex. 1 en C, ce qui est logique, et –1 en Basic, ce qui l'est beaucoup moins, cf. ch. II, §5, page 9).

8 - EXPRESSIONS

C'est un terme très employé en programmation, bien que rarement défini. Il faut croire qu'une telle notion est intuitive. On peut cependant tenter de définir une **expression** comme *un objet* ou, plus généralement, une *suite d'opérations* qui, après exécution, possède une valeur, cette valeur étant l'une de celles autorisées pour les objets du langage.

Les **instructions**, elles, représentent chacune un ordre ou une commande complète, un peu comme les *phrases* du langage courant, dont les *propositions* seraient comparables aux *expressions*. Les affectations constituent la majorité des *instructions* : on sait qu'elles comportent le signe **=** (ou le symbole **:=** dans la famille Algol, sauf en C). A droite du signe **=**, on peut trouver une expression quelconque ; mais à sa gauche, on ne peut placer qu'un nom de variable.

Les expressions possèdent un *type* : il leur est attribué par le logiciel *traducteur* à partir du type des objets présents dans l'expression. Le traducteur classe les objets par ordre de complexité croissante, caractères, puis entiers courts, puis flottants, ... selon le schéma de la figure 6-4, page 50. En général, une expression prend le type de son objet d'ordre le plus élevé. Après cette *montée* éventuelle dans l'ordre des types, il intervient, en cas d'**affectation**, un *changement de type*, complètement indépendant, dont nous avons parlé précédemment (§6, page 77). Par exemple, dans l'instruction Basic,

110 *perim***% = 2** ∗ *pi***#** ∗ *rayon*

où *perim* est une variable entière et *pi* un nombre en double précision, l'expression **2** ∗ *pi***#** ∗ *rayon* sera évaluée en double précision, le type de *pi*, l'emportant sur celui des autres objets de l'expression *(rayon*, n'ayant pas de type explicite, est un flottant simple par défaut). Par contre, sa valeur sera arrondie à un entier avant d'être affectée à la variable *perim***%** (on rappelle que, dans la plupart des autres langages, il n'y aurait pas arrondi, mais troncature à l'entier inférieur).

Un mot encore à propos de la **précision** des nombres. Il faut savoir que tous les calculs dans l'UC (surtout ceux sur les flottants), se font avec une précision bien meilleure que toutes celles mentionnées jusqu'ici. C'est seulement au moment de la sauvegarde en mémoire du résultat (affectation), ou bien au cours de certaines opérations (divisions entières, modulo, soustractions), que l'on peut faire chuter la précision comme indiqué ci-dessus (cf. chap. VI, §17, page 50).

9 - LES CONSTANTES

Comme leur nom l'indique, les constantes sont des entités dont la valeur est invariable tout au long du programme. Elles possèdent toutes l'un des types étudiés ci-dessus. Mais on ne les déclare qu'exceptionnellement, leur type se déduisant de leur écriture. Les conventions diffèrent assez peu d'un langage à l'autre.

Pour écrire une **constante entière** (simple), on n'emploiera aucune marque particulière. Sont par exemple des constantes entières :

2, **4095**, **0**, **–23**, **–32767**

Cependant, seront admises les formes octales, avec un zéro, un **&** ou les lettres **&O** en tête selon le langage, comme

&O176543, **0176543,**

égales à 64867_{10}, ainsi que les formes hexadécimales comme

&HF9B2, **0xF9B2**

égales à 63922_{10}, représentations possibles **(12)** pour les entiers positifs plus petits que 65537.

Les constantes **entières longues** sont suivies de **l** ou de **L**. Exemple :

2L, **–35000L**, **2000000L**

Quant aux constantes **flottantes** (ou réelles) en simple précision, elles sont obligatoirement caractérisées au moins par un point (la virgule, trop francophone, n'est pas admise), ou par la lettre **E** précédant la valeur de l'exposant (ou, en Basic, par le symbole terminal **!**). Il en va ainsi des constantes suivantes égales respectivement à 1, 100, 0.001 puis à π pour les trois dernières :

1., **1E2**, **0.001**, **3.1416**, **0.31416E1**, **31416E–4**

Les constantes réelles en double précision auront un **D** au lieu du **E** devant l'exposant ou bien (en Basic) seront terminées par le signe **#**, comme

–12D2, **0.1D–106**, **1200#**

Les constantes logiques, qui ne peuvent avoir que la valeur *vraie* (True) ou *fausse* (False), se notent, selon les langages :

TRUE, **FALSE**, **T**, **F**, **.T.**, **.F.**, **1**, **0**, **–1**, **0**

(12) Se souvenir que, dans un langage évolué, le nom d'une variable débute toujours par une lettre. Les formes hexadécimales ordinaires telles que **F9B2** seraient donc prises pour des variables.

Les constantes caractères s'écrivent avec un peu plus de diversité. Selon le langage, le nom d'un caractère imprimable doit être placé entre apostrophes ou entre guillemets pour être reconnu comme une constante caractère. Ainsi :

'A', **'?'**, **"y"**, **'ç'**

Les caractères ne figurant pas sur le clavier comme le caractère d'échappement ou ceux de commande ne peuvent être désignés que par des moyens détournés, souvent grâce à leur numéro ASCII (procédé d'ailleurs possible pour n'importe quel caractère). Ainsi le caractère échappement (ASCII n°27) pourra être désigné, selon les langages, par :

'\x1B', **'\033'**, **#27**, **27**

On ne peut pas vraiment dire que les constantes soient des objets du langage : souvent le *traducteur* ne leur affecte aucune mémoire, mais convertit les expressions où elles apparaissent en des séquences fixes d'instructions machine. Ces séquences comprendront ce qu'on a appelé des opérations *immédiates* (opérations avec des facteurs constants).

Signalons enfin que certains langages permettent de déclarer des **objets à valeur constante** sous une forme apparemment semblable à celle d'une variable. Cependant, le *traducteur* veillera jalousement sur un tel objet et interdira au programmeur d'en changer par la suite la valeur. Il en irait ainsi, à cause du mot-clé **CONST**, après la déclaration suivante :

CONST *pi* **= 3.1459 ;**

10 - LES COMMENTAIRES

Tout comme en langage machine, le programmeur peut émailler le programme-source de commentaires dont l'utilité est incontestable pour la relecture d'un travail. Ces textes sont repérés par des moyens variant beaucoup avec le langage.

En Fortran, toute ligne commençant par la lettre C placée **en première colonne**, sera considérée comme du commentaire. De même en Cobol, avec le caractère * .

En Basic, sera considéré comme du commentaire tout texte, jusqu'à la fin de la ligne, situé après le mot-clé **REM** (toutes versions), ou bien après l'apostrophe **'** (en GWbasic) ou après le point d'exclamation **!** (en Basic HP).

Dans la *famille point-virgule*, les commentaires peuvent être placés presque n'importe où. Ils sont encadrés par des caractères spécifiques : **{** ..... **}** en Pascal, **/*** .... ***/** en C).

11 - POUR RESUMER

Nous avons abordé dans ce chapitre des notions fondamentales, mais elles ne nous permettent pas à elles seules d'écrire un vrai programme informatique. On s'est limité à l'étude des instructions manipulant les objets les plus simples et à la façon d'introduire ces objets eux-mêmes.

On a défini l'*expression*, puis l'*instruction* et détaillé la plus importante de celles-ci, l'*affectation*, avec les problèmes qu'elle pose en cas d'un *changement de type* entre la variable affectée et l'expression située à droite du signe d'affectation.

On rappelle que les **variables** (qu'on peut considérer comme les **inconnues** du problème) possèdent un type, tenant compte de la constitution des unités centrales de calcul et de leurs mémoires. Les langages négligeant cet aspect ou ceux ne manipulant qu'un ou deux types d'objets sont peu performants. Les expressions elles-mêmes reçoivent implicitement un type pendant leur traduction et le programmeur doit en tenir compte s'il *mélange les types* dans une même expression.

Deux notions s'imposent à tout objet utilisé : tout d'abord, son type doit être clair (sauf dans quelques langages) ; il doit être défini par des conventions spéciales (Fortran) ou bien par une déclaration explicite et préalable. Deuxièmement, **tout objet employé dans une expression doit avoir été affecté** d'une valeur, souvent à l'aide de constantes, qu'on a appris à écrire malgré la diversité des formes pour certaines d'entre elles.

Ces notions doivent avoir été bien assimilées avant d'aborder le chapitre suivant, qui sera consacré à la manipulation des objets composites (tableaux, chaînes ...) ainsi qu'aux ruptures de séquences, indispensables à ce genre de manipulation. Nous terminerons avec les problèmes d'entrées-sorties, puis avec un aperçu des aides offertes à la programmation.

EXERCICES

1 - A quoi est égale l'expression **j * (i / j) + i MOD j**, si **i** et **j** sont des entiers? (si ce formalisme paraît trop abstrait, remplacer **i** et **j** par des nombres quelconques non divisibles entre eux).

Seuls des problèmes très simples, tout juste dignes d'une calculette, peuvent être résolus au stade où nous en sommes. Leur solution sera donnée, ce qui ne dispensera pas le lecteur de l'étudier avec le plus grand soin.

2 - **Problème de robinets**

On demande le temps nécessaire au remplissage d'une citerne cylindrique de diamètre 2,85 mètres et de hauteur 1,6 mètre, alimentée par un tuyauterie débitant 30 litres par minute, tandis que cette citerne fuit avec un taux de 150 litres par heure.

En GWBasic, on peut écrire la séquence ci-après :

```
10   diam = 2.85:   haut = 1.6                    ' dimensions en mètres
20   pi = 3.1416
30   vol = pi * diam^2 /4 *haut*1E3               ' volume en litres
40   fuite = 150 / 60                             ' fuite en litres / minute
45   alim = 30                                    ' en litres / minute
50   bilan = alim - fuite                         ' variable dont on pourrait se passer
70   temps = vol/bilan                            ' en minutes
80   h = temps\60:      min = temps MOD 60
85   nsec = (temps - INT(temps)) * 60
90   print "Temps de remplissage ="; temps; " mn ";
95   print " soit"; h;"h, "; min; "mn et "; nsec; "s."
100  END
```

Avec un problème aussi simple, on aurait pu écrire directement :

```
47  print "Temps= ", pi*diam^2*haut*250/(alim-fuite)
```

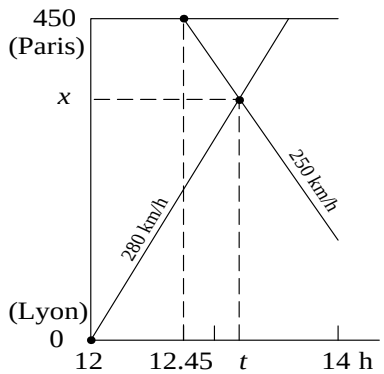
ou même

```
print  "Temps=", 3.1416*2.85^2 *1.6*250/(30-150/60)
```

On remarquera la *numérotation*, obligatoire, des lignes écrites en Basic. A la ligne 30, noter la succession des opérations : on tient compte de ce que, en l'absence de parenthèses, l'exponentiation, hautement prioritaire, se fait en premier ; puis vient le tour des multiplications et des divisions, les additions-soustractions se faisant en dernier. Notez une des façons d'écrire 1000 en flottant : il faut au moins un chiffre devant le E de l'exposant. Remarquez également l'emploi du symbole de division entière \ en ligne 80, de **MOD** (*modulo*) et de la fonction **INT** (qui donne la partie entière de son argument).

3 - Croisement de trains.

Un train part de Lyon vers Paris à midi tandis qu'un autre part de Paris vers Lyon à 12 h 45 mn. En adoptant 450 km comme distance Paris-Lyon, 280 km/h pour vitesse moyenne du premier train et 250 pour celle du second, calculer l'heure de leur croisement et la distance alors parcourue.



En algèbre, les équations ci-après donneraient la marche des trains, x étant la distance parcourue à l'heure t, v la vitesse moyenne, h l'heure de départ. La distance totale est x_0. Les indices 1 et 2 se réfèrent au premier et au second train :

$$x_1 = v_1 (t - h_1)$$
$$x_2 = x_0 - v_2 (t - h_2)$$

Les distances x_1 et x_2, comptées toutes deux à partir de Lyon, sont égales au moment du croisement *t*. De l'équation $x_1 = x_2$, on tire aisément la valeur de *t*.

$$t = (h_1\, v_1 + h_2\, v_2 + x_0) / (v_1 + v_2)$$

En Pascal, on pourrait alors écrire le petit programme :

```
PROGRAM   train_train ;
CONST
      x0  =  450.;                              { distance Lyon-Paris }
VAR
      x, h1, h2, v1, v2, t :    REAL ;
      h, min, sec        :    INTEGER ;
BEGIN
      v1  :=  280 ;      v2  :=  250 ;          { vitesses des trains }
      h1  :=  12 ;       h2  :=  12.75 ;        { heures  décimales }
      t  :=  (h1*v1 + h2*v2+x0)  /  (v1+v2) ;
      x  :=  v1 * (t-h1) ;
      h  :=  TRUNC(t) ;          t  :=  60*(t-h) ;
      min  :=  TRUNC(t) ;        sec  :=  ROUND (60*(t-min)) ;
      WRITELN  ('Réponse: ',h,'H ',min,'mn ',sec,'s.') ;
      WRITE   (' à la distance ', x:8:2, ' de Lyon.') ;
END.
```

Les mots-clés ont été ici écrits en majuscules, ce qui n'est pas du tout nécessaire en Pascal. Le mot-clé **PROGRAM** doit introduire le programme qui doit lui-même recevoir un nom (quelconque, mais sans espaces).

Des *sections déclaratives* suivent le titre : une section des *constantes* d'abord, dans laquelle on aurait pu placer toute grandeur ne devant pas subir de modification dans le calcul, comme **v1, v2, h1, h2.** Noter que **x0** sera prise comme une constante réelle à cause du point qui suit **450**. La section de déclaration des *variables* commence par **VAR** et comprend ici deux lignes, l'une pour les flottants simples (**REAL**), l'autre pour les entiers signés (**INTEGER**).

Le corps du programme proprement dit suit les sections déclaratives, il commence par le mot-clé **BEGIN** et se termine par **END suivi d'un point**. C'est un *bloc d'instructions*. Si le programme comprenait, comme c'est souvent le cas, d'autres blocs intérieurs au premier, ceux-ci auraient été délimités par les mots-clés **BEGIN** et **END** suivi du **point-virgule**. Remarquer le symbole **:=** comme opérateur d'affectation (sauf dans la section de déclaration des constantes) et le fameux *point-virgule* obligatoire à la fin de chaque instruction.

La conversion de l'heure décimale en heure sexagésimale a nécessité le recours à des mots-clés (ou plutôt à des *fonctions*) du langage : **TRUNC** qui donne la partie entière (ou tronquée) de son argument et **ROUND** qui en donne l'arrondi (l'une ou l'autre de ces fonctions est obligatoire en Pascal pour affecter une expression réelle à un entier).

L'écriture du résultat à l'écran utilise la fonction **WRITELN** (**LN** veut dire *line*), qui termine la ligne écrite par un *retour chariot* et un interligne. On verra dans le chapitre X le rôle du **format** dans l'instruction **WRITE** : ici le groupe **x:8:2** fait imprimer le contenu de la variable *x* avec 8 chiffres dont 2 décimales.

On notera également le double emploi de la variable **t**, qui désigne au début l'heure (décimale) du croisement et, à la fin, sa seule partie fractionnaire en minutes (nombre réel). Cette façon de faire est courante, bien qu'un peu gênante pour la *lisibilité* des programmes. Elle ne se justifie que lorsqu'il est nécessaire d'économiser la taille de la mémoire occupée par les variables. Ce double emploi est ici intentionnel : le lecteur devra bien assimiler que dans l'instruction **t := 60*(t-h);** le premier **t** reçoit le résultat du calcul de l'expression **60*(t-h)**, tandis que le deuxième **t** (en fait, le premier manipulé par l'UC) désigne encore l'instant de croisement en heures décimales calculé deux lignes plus haut. L'affectation n'a rien à voir avec une égalité algébrique (c'est pourquoi le Pascal, ultra-méticuleux, n'utilise pas dans ce cas le signe **=**).

4 - Récréation musicale.

La programmation en langage machine permet de faire exécuter des tâches variées à un ordinateur. Celle en langage évolué un peu moins. Voici cependant un exemple d'une possibilité peu connue du Basic, celle de faire jouer une musique rudimentaire par le petit haut-parleur de la machine.

On utilisera la fonction **PLAY** dont l'argument est une chaîne. Celle-ci contiendra les caractères suivants :

a, **b**, **c**, **...**, **f**, **g** pour les notes *la, si, do,..., fa, sol* ;

+ ou **–** , *altération*, qui, placée juste après une note, élève ou, respectivement, abaisse celle-ci d'un demi-ton ;

1, **2**, **4**, **8**, **16**... nombres *n* qui, placés après une note (et après son altération), indiquent sa durée 1/*n*, soit : 1⇒ronde, 2⇒blanche, 4⇒noire, 8⇒croche, 16⇒double croche ...

. point qui, placé après la durée, l'augmente de moitié ;

> ou < , qui élève ou abaisse d'une octave la suite de la mélodie ;

P*n* qui indique un silence de durée 1/*n* (1⇒pause, 2⇒demi-pause, 4⇒soupir, 8⇒demi-soupir, ...) ;

ainsi que des symboles généraux comme :

O*n* (jeu à la *n*ième octave),

T*n* (*tempo*, nombre de noires par minute),

L*n* (durée 1/*n* de toutes les notes de durée non précisée),

M*x* (*liaison* : **MN**⇒notes légèrement détachées, **ML**⇒notes liées, **MS**⇒notes bien détachées, symboles à placer avant le groupe concerné).

Ces symboles généraux sont actifs jusqu'à contrordre. Rien n'est prévu pour coder la *nuance* (l'intensité du son).

Remarque : la documentation du QuickBasic et celle du GWBasic indiquent que la suite **a,...g** correspond à notre gamme *do,...si*. Or, cette suite représenterait plutôt la gamme *la,...sol*. Il en est ainsi sur le PC de l'auteur et les deux airs programmés ci-après relèvent de cette convention. Le lecteur voudra bien vérifier, en écoutant la première ligne de ce programme, s'il en est de même sur son installation ; sinon, il lui faudra transposer la notation employée.

```
PLAY "p2  o2  t120  ML  cdefgab>c"                    ' gamme pour vérification
PLAY "p1."                                           ' silence de 3 demi-pauses

PLAY "t140  ML  o2  L8"                                      ' Il pleut bergère
PLAY "a >c4< a >c4< a f4. c4  P8  fef MN g4 g a2  P4  aga MN "
PLAY "b-4 b- ML >c4.< a P8 >cd MN c ML c <b- MN a ML a4. g4."
PLAY "p8 ga MN g g4 ML b- a4. >c4.< b- ag a4 f a4. g4. p16 "
PLAY " a >c4< a >c4< a b-4. >d4. p16 cdc< g4 a f2. "
PLAY "p1."
                                                             ' Les filles de la Rochelle
PLAY "T150  ML  o1  L8"
ch$ = ">d4 ec4.<ab >c<b4 p8 "
PLAY ch$ + ch$ + ch$ +  " g4 a MN b4 b b4 b >d4 d<  p16 ML "
PLAY "g4 a MN b4 b ML b a g a4 p8 "
PLAY "g4 a MN bbbbbb >d4 d<  ML p16  g4 a b>c<b a4 b g4 "
```

Chapitre X

LA PROGRAMMATION EN LANGAGE EVOLUE (deuxième partie)

1 - OPERATIONS SUR LES TABLEAUX

Les opérations permises sur les **éléments** d'un tableau sont *les mêmes* que celles autorisées sur les variables simples de même type. Il faut simplement savoir désigner correctement l'élément utilisé. Le *nième* élément du tableau TAB se désigne par **TAB(N)**, **tab[n-1]**, ... selon les langages, l'entier entre parenthèses ou entre crochets étant son indice. Le premier élément du tableau a pour indice 1 dans les langages anciens (Cobol, Fortran) et 0 dans la plupart des langages modernes ainsi qu'en langage machine [(1)].

La valeur de l'indice ne doit pas dépasser (déborder) celle autorisée dans la déclaration, c.-à-d., avec un tableau déclaré de taille *N*, *N*–1 en C et *N* dans la plupart des autres langages. Le **débordement** traduit toujours une erreur logique grave, mais de plus, bien souvent, il provoque la panne totale du calculateur (parce que le système ira modifier une case-mémoire hors du tableau réservé, case pouvant très bien contenir une information critique). Ce genre de dépassement est surveillé dans certains Basics très "protecteurs" (encore une disposition source de lenteur). Les autres langages offrent plus ou moins de protection, le débordement des tableaux n'étant en général contrôlé qu'à l'étape de la *compilation*.

2 - OPERATIONS SUR LES CHAINES

Dans certains langages comme le C, les chaînes ne sont que des *tableaux de caractères* et se manipulent comme les tableaux. On peut travailler sur chacun des caractères de la chaîne *ch* par exemple en l'appelant **ch[i]**, si *i* est son rang à partir de la gauche, le premier étant **ch[0]**. Il existe également des fonctions permettant un maniement global des chaînes.

Dans d'autres langages comme le Basic, seules les fonctions de traitement de chaînes en permettent la manipulation. On pourra presque toujours effectuer au moins les opération suivantes :

– extraction d'une **sous-chaîne** *(substring)*, qui peut être la partie droite, gauche ou médiane de la chaîne ; on doit alors préciser la taille de la sous-chaîne extraite (et sa position, si elle est en partie médiane) ;

– **recherche** de toutes les apparitions d'une sous-chaîne, ou d'un caractère, dans une chaîne donnée ;

– **transformation** d'une sous-chaîne ou d'un caractère en une autre chaîne, sa mise en majuscules (ou en minuscules), ... ;

– **concaténation** de deux chaînes (réunion bout à bout en une seule chaîne) ;

– **recopie** d'une chaîne dans une autre, ...

En Basic par exemple, les instructions ci-après *initialisent* (c'est-à-dire : *donnent une valeur à)* la chaîne **Ch1$**, puis en extraient les 5 premiers caractères :

```
Ch1$ = "L'apprentissage du Basic"
Ssch$ = LEFT$ (Ch1$, 5)
```

avec, pour résultat, l'affectation des 5 caractères "**L'app**" à la sous-chaîne **Ssch$** (**Ch1$** n'est pas modifiée). Si on écrit maintenant

```
MID$ (Ch1$,11,5) = "-sorcier"
```

la chaîne **Ch1$** est modifiée, mais le résultat dépend de la version Basic. En général, les 5 lettres à partir de la onzième, soit "**ssage**" sont remplacées par les 5 lettres "**-sorc**". **Ch1$** contient maintenant "*L'apprenti-sorc du Basic*". Parfois, c'est tout le mot **-sorcier** qui remplace la fraction **ssage** et la chaîne s'allonge. On peut connaître sa longueur réelle par :

```
long = LEN (Ch1$)
```

Les fonctions LEFT$, RIGTH$, LEN ... sont des *fonctions chaîne*. Si elles représentent elles-mêmes des chaînes, leur nom se termine par un **$**. Peu de langages manipulent les chaînes avec autant de simplicité que le Basic. En particulier, l'opérateur de concaténation est le signe **+** (il en va de même en Pascal). Par exemple,

---

(1) Dans certains langages, la valeur initiale de l'indice peut, avec une déclaration explicite, être un entier (simple) quelconque. Remarquer l'emploi, autour de l'indice tantôt de parenthèses, tantôt de crochets, selon le langage.

```
Q$ = "Salut":          Q$ = Q$ + " à toi."
Q := 'Salut';          Q := Q + ' à toi.';
```

transforme les chaînes **Q$** et **Q**, d'abord égales à "**Salut**", en "**Salut à toi.**". La première ligne est écrite en Basic, la seconde en Pascal.

Cette simplicité est due en partie à ce que les compilateurs actuels – ou les interpréteurs Basic – utilisent divers procédés pour délimiter la chaîne : il peut s'agir soit d'une marque spéciale à la fin de celle-ci (C), soit de l'inscription dans le premier octet (indice 0) de sa longueur réelle (Pascal, Basic). Les fonctions de traitement de chaînes opèrent caractère après caractère jusqu'à l'apparition de la marque ou jusqu'à ce que sa longueur soit atteinte. Ces procédés dispensent le programmeur de tenir un compte rigoureux des caractères présents dans sa chaîne ; il doit simplement veiller lui-même à ce qu'elle ne déborde pas de l'espace permis ou alloué lors de sa déclaration (attention aux concaténations).

En effet, si le système, au moment de l'*exécution*, connaît la longueur réellement occupée par la chaîne, il est rare qu'il en connaisse la taille *allouée* au moment de la compilation en réponse aux *déclarations* (cf. chap. IX, §5, p. 76). Il n'a pas de moyen alors d'empêcher son débordement sur des zones mémoire contiguës, ce qui provoque les mêmes catastrophes que dans le cas d'un débordement de tableau. Le Basic gère mieux les problèmes de débordement, mais le nombre d'instructions de contrôle ajouté pour cela rend ses programmes lourds et plus lents à exécuter.

Normalement, la chaîne Basic est **élastique** : elle s'allonge selon le besoin et s'adapte au nombre d'octets insérés par affectation ou concaténation. Seuls les Basic évolués permettent de déclarer des chaînes de longueur fixe (**MAP ...**, **DIM ... AS STRING ...**). Malgré cela, le Basic est insuffisant en matière de manipulation de chaînes.

Le langage C, extrêmement puissant en ce qui concerne le traitement des chaînes (car elles sont pour lui des tableaux de caractères), est également très tolérant envers les opérations sur elles. Par exemple, les **opérations arithmétiques** sur les caractères composant une chaîne, interdites dans la plupart des langages, sont permises en C (le caractère prend alors la valeur de son numéro dans la table ASCII). La manipulation des chaînes requiert en C beaucoup plus de vigilance de la part du programmeur qu'en Basic [(2)].

3 - BRANCHEMENT (GOTO)

Tout comme un programme en langage machine, un programme en langue évoluée se déroule *en séquence*, sauf ordre contraire. Ces ordres contraires introduisent des ***ruptures de séquence***. Ces dernières existent aussi en langage machine (ce sont principalement les *sauts*, cf. chap. VII, §3, page 54). Les sections suivantes leur sont consacrées.

La rupture la plus simple est le **branchement inconditionnel**, en général introduit par l'instruction

GOTO *étiquette*

L'étiquette revêt des formes diverses. En GWBasic (ou en Basica), elle ne peut être que le numéro de la ligne sur laquelle on désire dérouter le programme. Dans des langages comme le Fortran ou le Pascal, cette étiquette est un numéro par lequel on doit référencer l'instruction de déroutement. Ce numéro n'a aucune autre signification et ne joue aucun autre rôle que celui d'assurer ce branchement.

Les autres langages (ainsi que les Basic *haut de gamme*) acceptent comme étiquette un mot *littéral*, bien plus significatif qu'un numéro. Tel est également le cas – on l'a vu – du langage machine symbolique.

Si le Pascal n'offre pas cette facilité, c'est qu'il affiche une aversion profonde pour le **GOTO**. Il est certain que les innombrables **GOTO** indispensables avec les *Fortran* primitifs et les *Basic* ordinaires rendent la relecture de ces programmes difficile. Des auteurs ont même qualifié le **GOTO** de *monstre antédiluvien*. Il n'en est pas moins vrai que, dans des cas très particuliers, le **GOTO** est l'instruction la plus indiquée.

L'instruction **GOTO** *étiquette* est souvent précédée d'une condition, introduite par un **IF**. Le branchement ainsi programmé, très proche du langage machine, s'appelle **branchement conditionnel**.

Quelques langages plutôt anciens utilisent le **GOTO calculé** (ou *aiguillé*). Plusieurs étiquettes numériques suivent alors le mot **GOTO** , puis une expression. Le branchement se fera à la *n*ième étiquette, si *n* est la valeur de l'expression au moment du branchement. Par exemple :

```
GOTO 350, 100, 200, j
ON j GOTO 350, 100, 200
```

permettront de se brancher à l'étiquette **350** si *j*=1, **100** si *j* =2, **200** si *j*=3. Si *j* prend une autre valeur, on continue en séquence (on passe à l'instruction suivant le **GOTO**). Souvent, *j* peut être une expression.

Ces moyens d'aiguillage, plutôt lourds, sont remplacés, dans les langages modernes, par les procédés décrits ci-après.

(2) Ceci est dû en particulier à l'usage intensif de pointeurs dans le traitement des chaînes en C. Si le débordement de celles-ci n'entraîne le plus souvent qu'une erreur localisée et détectable, les erreurs sur les pointeurs déclenchent presque aussitôt le *plantage* (l'arrêt complet) du système.

4 - INSTRUCTION IF , BLOC D'INSTRUCTIONS

Chaque langage offre la possibilité de poser des conditions avant l'exécution des instructions. On fait précéder l'instruction (ou plus généralement le bloc d'instructions) par le mot-clé **IF** suivi d'une expression logique. On a alors le schéma suivant :

IF *(expression logique)* **THEN** *bloc d'instructions*
ELSE *bloc d'instructions*

L'instruction **IF** la plus générale prévoit plusieurs cas. **SI** *expression logique* est vraie, **ALORS** le premier *bloc d'instructions* est exécuté; **AUTREMENT** (si elle est fausse), c'est le deuxième qui l'est. On peut dire que l'expression logique est la **condition** d'exécution de l'un ou l'autre bloc.

Le schéma ci-dessus peut subir quelques variantes : l'*expression logique* est soit délimitée par des parenthèses, soit par le mot-clé **THEN**, une seule de ces deux formes étant nécessaire dans un langage donné. D'autre part, la partie **ELSE** n'est pas obligatoire, elle n'est même pas possible avec certains langages. Enfin, des lignes

ELSE IF (*expression logique***) THEN** *bloc d'instructions*

peuvent s'intercaler entre les deux précédentes ou suivre un **IF** isolé. La compréhension de leur mécanisme n'est pas trop difficile : il ne peut pas y avoir exécution à la fois du **IF** et du **ELSE** (ou **ELSE IF**) qui lui correspond ; d'autre part, le **ELSE** se rapporte au dernier **IF** ou **ELSE IF** (à moins de délimitations contraires explicites au moyen de *blocs*).

L'expression logique peut être composite, c.-à-d. formée de plusieurs expressions logiques simples reliées par des opérateurs **AND**, **OR**, **NOT** et parfois **XOR** (cf. chap. II, p. 10). Enfin des groupes **IF ... THEN** peuvent être imbriqués les uns dans les autres.

Il nous faut définir ici ce qu'est un **bloc**. C'est une suite d'instructions encadrées par des *marques de bloc* (particularité des langages de type Algol). En Pascal, le bloc est délimité par les mots **BEGIN** et **END** :

BEGIN *instruction-1* **;** *instruction-2* **; .... ; END ;**

En C, on dispose du même schéma, à ceci près que les mots-clés **BEGIN** et **END** sont remplacés par les accolades **{** et **}**. En GWBasic, on peut également parler de bloc: il commence au **THEN** et se termine à la fin de la ligne *(bloc monoligne)*. Dans certains Basic, on peut écrire des blocs multilignes encadrés par les mots-clés **THEN** et **ENDIF** .

Il est vivement recommandé aux programmeurs de décaler de 2 ou 3 colonnes toutes les instructions d'un bloc donné afin de les identifier plus facilement (procédé appelé *indentation*).

5 - INVENTAIRE DE CAS

Les langages de type Algol permettent de présenter de façon très claire un inventaire d'options parmi lesquelles le programme choisira selon la valeur que prendra une expression. On définit avec **SELECT** ou **CASE** l'expression ou la variable clé, puis on cite tous les cas d'intérêt, chacun étant suivi du *bloc d'instructions* à exécuter dans cette éventualité. On peut terminer l'énumération par une ligne donnant la conduite à tenir quand aucun des cas cités ne se réalise.

Par exemple, à une question l'opérateur devrait répondre en frappant **O** (pour oui) ou **N** (pour non). On programme ainsi ce choix en Pascal :

```
CASE c OF          {c est le caractère frappé}
'O', 'o' :    BEGIN   bloc d'instructions n° 1  ; END ;
'N', 'n' :    BEGIN   bloc n° 2                 ; END ;
ELSE beep ;
END ;
```

où **beep** serait une fonction émettant un son.

Les expressions ou variables examinées ne peuvent avoir que des valeurs entières simples ou bien être des caractères. Mais certains Basic (HP, PowerBasic, ...) sont sur ce point encore plus puissants : ils acceptent des cas d'inégalité dans l'énumération. Par exemple, pour résoudre une équation du second degré ax^2+bx+c, on programmerait ainsi la conduite à tenir selon la valeur du discriminant $\Delta = b^2-4ac$:

```
50 SELECT CASE delta          ' delta = discriminant
60 CASE = 0 :      x1 = -b/(2*a) :      x2 = x1
70 CASE > 0 :      r = SQRT(d) :
                   x1=(-b+r)/(2*a):
                   x2=(-b-r)/(2*a)
80 CASE ELSE: PRINT "Solutions non réelles."   'Δ<0
90 END SELECT
```

Dommage que les autres langages n'aient pas cette facilité. Bien sûr, on peut résoudre ce problème avec des **IF** successifs :

```
IF delta=0 THEN BEGIN x1:=-b/(2*a) ; x2:=x1 ; END
 ELSE IF delta>0 THEN BEGIN
                      r := SQRT(delta) ;
                      x1 := (-b+r)/(2*a) ;
                      x2 := (-b-r)/(2*a) ;
                      END
 ELSE WRITELN ('Solutions non réelles') ;
```

Ces lignes sont en Pascal. Noter l'*indentation* du bloc multiligne ainsi que l'absence de point-virgule avant les **ELSE** associés aux **IF** .

6 - LES BOUCLES

On peut dire qu'une boucle obéit au schéma général ci-après :

en-tête bloc d'instructions final

Le bloc d'instructions sera exécuté alors *n* fois, le **nombre de pas** *n* (ou *itérations*) étant déduit de l'*en-tête* ou du *final*.

a - Dans la boucle de type **FOR** ou **DO**, on donne dans l'en-tête le nom *var* de la variable de boucle, sa valeur initiale i_0, sa valeur limite i_m et l'accroissement *incr* qu'elle subira à chaque pas. En Basic, le *final* consiste dans le mot-clé **NEXT** suivi ou non du nom de la variable de boucle. En Fortran, le *final* est l'instruction précédée de l'étiquette *etiqu* qui suivait le mot-clé **DO** (cette instruction fait encore partie de la boucle). En Pascal et en C, le final consiste simplement dans la marque de fin de bloc sous la boucle. On trouve les formes suivantes :

FOR *var* = i_0 **TO** i_m **STEP** *incr* **.....** *instructions* **.....** **NEXT** *var*	Basic
DO *étiqu var* **=** i_0 , i_m , *incr* **....** *instructions* **....** *étiqu : dernière instruction*	Fortran

FOR *var* = i_0 **(DOWN) TO** i_m **DO**
BEGIN ... *instructions* **... ;**
END ;
	Pascal (incrément = ±1)

for (***bloc des conditions* **) **{ ...** instructions **... ; }**	C

Nous ne détaillerons pas le bloc des conditions multiples, puissantes et complexes du C. En Cobol, le mot-clé pour les boucles est **PERFORM**. Il faut remarquer que dans la majorité des boucles, l'incrément de la variable est 1 ; il est alors inutile de l'écrire (**STEP**, *incr*, ... sont omis). En Pascal, il est forcément implicite et ne peut être que 1 (mot-clé **TO**) ou -1 (mot-clé **DOWN TO**). Enfin, ajoutons que les marques de bloc en C ou Pascal peuvent être omises si ledit bloc ne comprend qu'une instruction.

L'exemple suivant en Basic montre comment on procède à un calcul itératif : il s'agit de trouver le nombre de jours *Nbj* écoulés entre la bataille de Marignan (14 septembre 1515) et le 1er janvier 1991 :

```
10 Nbj = 108                        ' jusqu'à la fin de 1515.
20  FOR N% = 1516 TO 1990           ' incrément 1 par défaut
30  Nbj = Nbj + 365
40   IF ((N% MOD 4 = 0) AND (N% MOD 100 <> 0))
      THEN Nbj = Nbj + 1            ' années bissextiles
50  NEXT N%
60 PRINT "Nb de jours: "; Nbj
```

On aurait pu donner le type réel à **N** en l'écrivant **N** et non **N%**. Mais le calcul est plus rapide avec **N** entier (le gain serait de 30 à 100 si **Nbj** avait pu lui aussi être un entier, mais il dépasse la valeur 32 767).

b - Les boucles de type **WHILE** ou **REPEAT** sont très proches du langage courant. On pourra trouver des formes comme celles-ci:

WHILE *expr_logique* **...** *instruct.* **... WEND**	Basic
REPEAT ... *instruct.* **; ...** **UNTIL** *expr_logique* **;**	Pascal
WHILE *expr_logique* **DO** **BEGIN** *instruct.* **; END ;**	Pascal
do { ***instruct.* **; } while (*expr_logique***) ;**	C
while (***expr_logique*) { ***instruct.* **; }**	C

Le bloc d'instructions *instruct.* est exécuté *n* fois, *tant que* (avec **WHILE**) l'*expr_logique* reste vraie, ou bien au contraire *jusqu'à ce qu'elle* devienne vraie (avec **UNTIL**). Bien sûr, il faut faire varier quelque chose dans le bloc pour que ces calculs répétitifs soient utiles.

La boucle est très bien adaptée au calcul sur les **tableaux**. Leurs éléments ne se distinguant les uns des autres que par un indice, il suffit de faire varier celui-ci pour que la boucle exécute son travail (3) sur tous ses éléments. Il en va de même avec les chaînes.

Cet outil nécessite tout de même quelques précautions, des divergences se manifestant selon les langages. Par exemple, il faut savoir que parfois des boucles s'exécutent toujours au moins une fois, même si la condition d'itération n'est pas satisfaite dès le départ. C'est le cas des boucles de type **do ... while** et **repeat ... until** puisque leur condition n'est examinée qu'à la fin de l'exécution. C'est parfois, paradoxalement, le cas de certaines boucles de type **DO** .

Une autre difficulté peut surgir à propos des sorties prématurées de boucle. Alors qu'il est toujours interdit d'*entrer* dans une boucle autrement que par sa première instruction (son *en-tête*), on peut en sortir, grâce à un **IF** par exemple, avant sa fin normale. Mais, pour ce faire, certains langages obligent le programmeur à se placer d'autorité dans les conditions interdisant la poursuite de l'itération, par exemple en donnant à la variable de boucle une valeur supérieure à la valeur maximum. Dans d'autres (Basic), on ne peut en sortir que par les instructions **EXIT DO**, **EXIT FOR** ...

Terminons le sujet par le problème de l'**imbrication des boucles**. En effet, toute boucle peut en contenir une autre (on parle alors de boucles à plusieurs niveaux), mais la boucle intérieure doit toujours être entièrement contenue dans celle de niveau inférieur. Leur chevauchement n'est pas admis. Là encore, l'écriture avec *indentation* est fort utile, car elle permet de détecter les manquements à cette règle.

(3) Les sommations et produits multiples mathématiques, représentés par les symboles Σ et Π, sont traduits par des boucles de type *a*.

7 - SOUS-PROGRAMMES ET FONCTIONS

On a vu dans le chapitre VI (p. 49) ce qu'était un sous-programme : il effectue un travail sur des objets du programme principal, ou, plutôt, du programme l'ayant appelé. La **fonction** est très voisine. Plus modeste, elle ne travaille souvent que sur un ou deux objets du programme appelant et **prend** elle-même, après traitement, une valeur égale à son résultat. Elle correspond bien à la *fonction* ou *application* mathématique : pour un argument *x*, elle prend la valeur *y(x)*. Prenons l'exemple d'une fonction, qu'on appellera **puiss (x,m)** qui calcule x^m. Dans le programme appelant, si on a besoin de $a^2 + b^2$, on pourra écrire

```
som_quadr := puiss(a,2) + puiss(b,2) ;
```

La fonction *puiss* pourra être ainsi définie (en Pascal) :

```
FUNCTION puiss (x : REAL ; m : INTEGER) : REAL ;
VAR i : INTEGER ; y : REAL ;
BEGIN
  IF x=0 THEN y := 0
       ELSE BEGIN
       y := 1 ;
       IF m<0 THEN FOR i:=1 TO -m DO y := y/x ;
       IF m>0 THEN FOR i:=1 TO m DO y := y*x ;
       END ;
puiss := y ;
END ;
```

Dans la première ligne, les variables *x* et *m* constituent la **liste de définition** ; ce sont des *variables muettes*. On remarquera la correspondance entre la variable *a* (qui doit être un réel) dans le programme appelant et *x* dans la fonction, ainsi qu'entre 2 et *m* (tous deux entiers). C'est au moment de l'*appel* que les vraies valeurs, celles de la *liste d'appel*, *a* et 2, se substituent à *x* et *m*. Contrairement à l'impératif énoncé page 77, chapitre IX (relatif au programme principal), **il ne faut pas initialiser** les variables figurant dans la liste de définition d'une fonction ou d'un sous-programme (SP). Ce procédé est très général.

Les SP fonctionnent de la même façon, à ceci près qu'ils ne prennent pas (ne *retournent* pas) eux-mêmes une valeur. Ils sont souvent de plus grande envergure et comportent une liste d'échange plus longue. La différence entre fonction et SP est donc faible, à tel point que le C ne fait aucune distinction entre eux.

Dans un programme en C, le simple fait de *citer* une fonction (de l'*invoquer* en franglais) l'appelle ipso facto. Il en est de même pour les SP dans la famille Algol. Mais en Fortran ou en langage machine (parfois en Basic), il faut faire précéder leur nom du mot-clé **CALL**. Le SP ou la fonction se terminent à la fin du bloc qui les constitue (famille Algol) ou bien avec le mot-clé **RETURN**. Après exécution des SP ou des fonctions, on revient au programme appelant, à l'instruction ou à l'expression située juste après celle d'appel.

Les SP s'appellent **procédures** en Pascal. Tout langage qui permet leur emploi est qualifié de *procédural*.

On a vu que fonction ou SP effectuent un traitement sur les variables ou objets du programme l'ayant appelé. L'échange des valeurs est assuré par la mise en correspondance de la liste d'appel avec la liste de définition. Cependant, il existe un deuxième moyen, cité dans le chap. VII, §11, p. 59 : il s'agit de l'emploi de variables **communes** ou **globales**. La *portée* de ces variables s'étend non seulement au programme principal, mais à l'ensemble des SP et des fonctions.

En Fortran, de telles variables sont déclarées grâce au mot-clé **COMMON**. Dans la famille Algol, pour que des variables soient globales, il suffit de les déclarer hors de tout bloc. Il n'y a pas besoin de citer ces variables dans la liste d'appel : elles sont connues du SP comme du programme appelant et ont le même nom partout. Ces variables peuvent être utilisées, modifiées, par n'importe quel programme, SP ou fonction. C'est un procédé qui peut paraître simple, mais qui est peu pratique : il ne permet pas l'écriture séparée des SP, ni un réemploi aisé de SP déjà mis au point, alors que la technique de la liste d'appel assure une très grande indépendance entre les noms employés dans le programme principal et ceux employés dans les SP.

Le Basic *classique* – c'est là une de ses faiblesses – ne connaît que des variables globales. Aussi, ses SP ne sont-ils pas de vrais SP. Appelés par le mot-clé **GOSUB** suivi du numéro de la ligne où ils débutent, ils effectuent un traitement à l'écart du programme principal, mais avec les mêmes variables. On revient au programme appelant lorsqu'on rencontre le mot-clé **RETURN** (ce qui différentie le **GOSUB** du **GOTO**). Ce procédé est toutefois bien utile pour de petits traitements répétitifs. Quant aux *fonctions*, leur définition ne peut occuper plus d'une ligne en Basic classique, ce qui est plutôt restrictif ; cette ligne doit commencer par **DEF FN** ; à part cela, ces *fonctions* se comportent comme celles des autres langages (indépendance des noms, correspondance par listes). Les Basic évolués sont dotés de vrais SP (avec indépendance des noms).

Une autre classe de rupture de séquence concerne les **débranchements sur erreur**. Dans certains langages, les fonctions prennent une valeur typique si une erreur s'est produite pendant leur exécution. On l'examine après les appels où une erreur est prévisible. En Basic, on peut programmer la gestion des erreurs grâce à une variable très utile, **ERR**, qui prend une valeur non nulle, en rapport avec l'erreur, s'il s'en est produite une. On enclenche ce processus avec l'instruction **ON ERROR GOTO** *n* , avec, à l'étiquette *n*, un SP comportant plusieurs branches commençant par **IF ERR = ... THEN ...** où l'on développe la conduite à tenir pour chaque erreur prévisible.

8 - LES ENTREES

Le programme doit communiquer de temps à autre avec l'opérateur ou avec *l'extérieur*. Cet échange s'opère grâce aux périphériques et surtout grâce aux deux principaux, l'écran et le clavier.

Les langages des temps héroïques (disons en gros avant 1970) ignoraient ces deux périphériques, alors peu répandus. L'entrée de l'information passait par le lecteur de cartes et la sortie par l'imprimante. Le rôle de ces organes est à peu près le même que celui de l'écran et du clavier, à ceci près que ces derniers permettent aujourd'hui une forme de communication autrefois impossible, le dialogue en *temps réel* (on l'appelle également l'*interactivité*).

L'introduction "manuelle" (ou **entrée**) des *valeurs* pour les variables (on les appelle *les données*) peut être prévue dans le programme par des instructions débutant avec des mots-clé tels que **INPUT**, **ENTER**, **READ**, suivis d'un nom ou d'une *liste* de noms de variables. Une pause s'ensuit ; un message doit avertir l'opérateur de cet arrêt. On frappe alors la ou les valeurs à donner à ces variables. Par exemple, les instructions Basic :

```
INPUT "Donnez valeur de A et de B :", A, B
INPUT "On continue ? (Oui ou Non)", repons$
```

devront être suivies de la frappe au clavier respectivement de deux valeurs numériques, puis d'au moins un caractère. Un retour-chariot doit suivre la frappe de la valeur de **B**, puis celle de la valeur donnée à **repons$**. Quand la liste est *satisfaite* (c.-à-d. quand toutes ces variables ont reçu des valeurs acceptées par le système), le calcul reprend. Mais le programmeur sera sage de prévoir la plus grande fantaisie dans les réponses, y compris les erreurs de bonne foi. Il aura à cœur de les analyser et d'en vérifier le bien-fondé. Bien souvent, on demande à l'opérateur de frapper **O** pour *oui* et **N** pour *non*. Encore faut-il prévoir aussi bien la frappe d'une majuscule que d'une minuscule (voir le premier exemple du **CASE**, §5 p. 87).

Tous les langages prévoient *l'entrée* de données à la console par des instructions de ce type, mais en général en en décrivant le *format* (voir §9). En Basic, il existe une autre possibilité, très simple, celle du couple **READ-DATA** : quand le programme rencontre un **READ** suivi du nom de quelques variables, il leur affecte des valeurs prises dans les listes précédées du mot **DATA**. En fait, chaque variable reçoit la valeur de la donnée sur laquelle pointe une sorte de *curseur* virtuel. Ce curseur avance d'une donnée dans les listes **DATA** à chaque affectation de variable dans une liste **READ**. Ce procédé permet ainsi l'exécution successive d'un même programme avec des valeurs différentes, prévues dès la programmation. Voici un exemple :

```
DATA 1990, "avril"
READ annee, mois$, quantiem
DATA 30, "lundi", 1789, "juillet", 14, "machin"
READ nomjour$
```

Seront affectés successivement la valeur **1990** à *annee*, la chaîne **avril** à **mois$**, **30** à **quantiem** et **lundi** à **nomjour$**. Si l'on n'avait pas respecté la concordance entre les données-chaîne et les variables-chaîne, l'interpréteur aurait signalé une erreur. Tout se passe comme s'il n'y avait qu'une seule liste **DATA** et une seule liste **READ** puisque leur position dans le programme est sans importance (seule exception à cette règle : l'instruction **RESTORE** ramène d'autorité le curseur virtuel en tête du premier **DATA** qui la suit). Si l'on exécute une deuxième fois le programme, les valeurs affectées seront **1789**, **juillet**, **14**, **machin.**

Tous les langages permettent de lire des données dans un *fichier* écrit sur un *volume* quelconque, à l'aide de mots-clé voisins des précédents. Il est toujours plus difficile d'aller lire dans un fichier que sur la console. Il faut d'abord *ouvrir* le fichier, c.-à-d. faire correspondre avec le nom de ce fichier un pointeur ou bien un numéro (qui sera un indice dans un tableau de pointeurs). On vérifiera toujours que cette *ouverture* s'est bien déroulée et que le système a trouvé le fichier. On lira ensuite les données *en séquence* logique, i.e. dans l'ordre où elles ont été écrites, sans se soucier de l'éparpillement physique du fichier (le système se gèrera ce travail). Cependant, il faudra en général expliciter le *format* (la forme) sous lequel les données sont écrites.

On appelle *fichier texte* ou *ASCII* un fichier écrit avec des instructions proches de celles d'impression. Toute donnée sera écrite en ASCII, y compris les nombres, figurés par des suites de caractères (ch. VI, §3) comme on les écrirait "à la main". Un *fichier binaire* (ou *non formaté*) est un fichier écrit en langage machine. Les fichiers texte sont plus encombrants, plus longs à lire et écrire (avec un *format* explicite), mais ils sont exploitables par de nombreux logiciels, surtout si l'on conserve le même système d'exploitation. Par contre, les fichiers binaires ne peuvent être lus (sans *format*) que par des programmes respectant les mêmes conventions d'écriture en LM ; il est très difficile de les échanger entre des programmes écrits dans des langages différents.

Il existe une possibilité de lire des données dans un fichier **directement**, c.-à-d. sans lire toutes celles qui précèdent. Il faut que le fichier ait été au préalable agencé par le système ou par le programme en *enregistrements* de longueur fixe. Ces fichiers d'accès direct sont d'un usage beaucoup moins fréquent que les précédents, dits d'*accès séquentiel*.

9 - LES SORTIES

On appelle *sortie* toute émission vers l'extérieur de la machine de messages communiquant le résultat d'un calcul ou d'un traitement. Cette émission est provoquée dans le programme par une instruction du type **WRITE**, **PRINT** ou **OUTPUT ...** Par exemple :

```
PRINT "La quantité a = " ; A ; " unités."
```

(en Basic) affichera à l'écran le texte "**La quantité a =**", suivi de la valeur de la variable *A*, puis du mot "**unités.**" Les éléments de cette ligne seront séparés par un seul espace, à cause du point-virgule entre arguments dans la *liste* **PRINT**. Si, dans cette liste, on avait mis des virgules, l'espacement aurait été très grand (tel qu'on obtienne au plus 4 arguments par ligne).

Tous les langages permettent au concepteur de définir la *forme* que doit revêtir la *valeur* des variables. On dit qu'on fixe le **format** de sortie : on peut ainsi préciser combien de chiffres on désire après la virgule, combien d'espaces entre les valeurs... (ou bien si on veut placer des astérisques ou des zéros devant les nombres, comme sur les chèques informatisés), ou revenir à la ligne, ... et bien d'autres choses encore. Par exemple, en supposant que la variable *PI* soit égale à 3,14159 (π), les instructions Basic suivantes

```
PRINT PI :
PRINT USING "##.##" , PI :
PRINT USING "**##.###" , PI :
PRINT USING ".##" , PI :
```

font afficher respectivement **3.14159**, **3.14**, *****3.142** et **%3.14** (le signe **%** signale que le format "**.##**" s'est avéré insuffisant). Les chiffres affichés ou imprimés viennent remplacer les symboles littéraux du format (les **#** et autres). Les langages autres que le Basic autorisent une gestion très fine des formats. Le Cobol utilise pour ce faire les descripteurs **PIC** ou **PICTURE**. Si un format manque dans une instruction de sortie, le langage en utilise un par défaut, rarement esthétique. Il peut arriver que le format ne permette pas d'écrire la donnée. Le comportement du système diffère alors selon le langage. Avec certains (Basic), il n'écrit rien ou bien des symboles alertant l'utilisateur de la non-conformité du format. Avec d'autres, il remplace le format fixé par le format par défaut. Avec certains langages peu protecteurs, on peut même obtenir une sortie totalement erronée (en C par exemple).

C'est en partie au soin apporté à la présentation des *sorties* qu'on reconnaît un bon programmeur. Hélas, les symboles des formats varient d'un langage à l'autre et le programmeur multilangage les confond souvent, s'il n'en garde pas la liste à portée de main.

L'écriture sur papier avec l'imprimante s'obtient par des ordres similaires, souvent par les mêmes, mais après avoir *redirigé* la sortie, c.-à-d. définie celle-ci comme étant non l'écran, mais l'imprimante. Les écritures sur **disques, disquettes** ou **bandes** sont d'autres formes de sortie qu'on programmera, avec ou sans format, selon l'étude de la section précédente. L'ouverture du fichier en écriture est un peu plus délicate. Sans précautions, on peut ouvrir un fichier déjà existant et l'écraser, c.-à-d. surcharger ses données ; pour éviter cet ennui, il faut étudier après l'ouverture, selon le langage, soit la variable d'erreur appropriée, soit la valeur retournée par la fonction **OPEN** .

10 - BIBLIOTHEQUES

Le Basic comporte un grand nombre de mots-clés permettant entre autres le calcul direct de *fonctions mathématiques* évoluées (trigonométriques, exponentielle, logarithme ...) ainsi que le *traitement* des chaînes et même le dessin, ce qui lui vaut d'être doté, dans ses versions avancées, de plus de 400 mots-clés ! Le Cobol, de même, en possède près de 500.

La plupart des autres langages procèdent autrement : peu de mots-clés (moins de 40 par ex. en C), ne comprenant pas les fonctions ci-dessus, qui sont regroupées dans des *bibliothèques* annexées au compilateur. Ce procédé procure à la fois plus de souplesse (on peut aisément récrire ces fonctions) et plus de rigueur : le noyau des instructions indispensables, réduit au strict minimum, peut être beaucoup plus facilement normalisé. Revers de la médaille, ces bibliothèques dépendent du concepteur du *langage* et ne sont donc pas forcément compatibles entre versions d'origines différentes.

En sus des fonctions mathématiques, on trouve dans les bibliothèques des *fonctions d'entrée-sortie* (que le *noyau* du langage n'assure pas toujours), *fonctions graphiques* (permettant de réaliser du dessin), fonctions de *traitement de chaîne*, de *gestion des fichiers*, ... On conçoit que la machine doive posséder un moyen d'en trouver le contenu quand elles sont appelées dans le programme. Parmi les méthodes employées, la plus courante consiste à utiliser un autre logiciel, souvent appelé **éditeur de liens** (ou *link editor*, mais qui répondrait bien mieux au nom d'*assembleur*). On lui fournit en entrée le *programme compilé* ainsi que les bibliothèques renfermant les fonctions citées dans le programme. Le logiciel créera les liens nécessaires à l'exécution correcte de ces fonctions (on dit qu'il *résout* les *références*). Le programme compilé s'appelle souvent *module objet* et celui comportant les liaisons avec les bibliothèques *module exécutable*. C'est en effet seulement à ce stade qu'il pourra être chargé en mémoire et exécuté.

11 - AIDES LOGICIELLES A LA PROGRAMMATION

La rédaction du programme – ou *développement* – consiste à écrire une suite d'instructions dans un fichier à l'aide d'un *éditeur* (cf. chap. IX, page 74). Revenant sous le *système d'exploitation*, on soumet ce fichier au **compilateur** ou à l'**interpréteur**, ce qui exige entre autres de changer de *répertoire* ; ensuite, sauf en Basic, il faut présenter à l'**éditeur de liens** le fichier résultant accompagné des bibliothèques utilisées, puis revenir sous le *système* pour faire exécuter ledit programme et enfin aller lire les résultats ou la liste des erreurs dans les fichiers ad hoc. Cette gymnastique est bien simplifiée si on emploie des fichiers de commandes tels que les fichiers **batch** du DOS décrits dans les exercices page 71. Par exemple :

clex *progr*

déclenchera l'exécution successive des opérations mentionnées si le fichier **clex.bat** comporte la liste des commandes nécessaires avec un procédé de passage d'arguments. Le nom de la commande rappellera utilement son rôle, comme ici ceux de **C**ompilation, création de **L**iens, **EX**écution.

Cette procédure est malgré tout délicate, à cause des nombreux paramètres à transmettre d'une phase à l'autre et des erreurs à prévoir à chaque phase. Tout est bien plus simple en Basic : sous l'éditeur Basic, on tape **run** ou bien on appuie sur la touche de fonction F2 prévue pour écrire **run** à la place de l'opérateur. On peut également revenir sous le système d'exploitation et taper **run** *fichier* (le SE doit posséder un moyen pour trouver ce fichier, par exemple en rendant son répertoire actif) **(4)**.

Les firmes vendant des logiciels de programmation offrent souvent des **systèmes intégrés**, permettant un accès rapide par fenêtres et menus aux différentes composantes du logiciel et du système : l'*éditeur*, le *compilateur*, le *link editor*, le *chargeur*. De plus, on disposera d'un procédé de pointage des erreurs très convivial, ainsi que d'un outil de mise au point rationnel. Ces logiciels sont d'un tel confort et d'une telle puissance qu'il ne faut pas hésiter à se les procurer **(5)**.

Il reste pourtant encore beaucoup à faire pour que ces logiciels soient parfaits : en particulier, leurs messages d'erreur sont souvent peu clairs ou inappropriés. Malgré cela, la rapidité d'analyse des erreurs fait que l'utilisateur soumet très vite son projet au *système*, bien avant d'être sûr de sa validité. L'interactivité encourage la paresse du programmeur.

L'informaticien écrivant un programme interprété (comme en Basic) apprécie beaucoup le fonctionnement en **mode pas à pas**, avec sa possibilité de lire après chaque instruction la valeur de n'importe quelle variable. Aussi, les *systèmes intégrés* aux compilateurs, pour ne pas être en retrait sur les interpréteurs, offrent également, pendant la phase de mise au point d'un programme *(debugging)*, la possibilité de travailler en pas à pas, avec affichage de la valeur prise par les variables. Bien sûr, ce procédé ralentit considérablement l'exécution, mais cela n'a pas d'importance dans cette phase. Ces *debuggers (dévermineurs*, et pourquoi pas *épouilleurs* ou *détecteurs* ?) permettent une recherche rationnelle et rapide des erreurs **(6)**.

On aura remarqué que tous les langages, malgré leur diversité, assurent en gros les mêmes fonctions. La simple lecture de ces deux chapitres ne saurait suffire à former un programmeur. L'étude approfondie du langage choisi sera toujours nécessaire. Il faudra apprendre la *portée* et les limites de chaque *opérateur* et de chaque *mot-clé* : on ne programme pas dans l'à-peu-près. Certains langages sont très *protecteurs* (Pascal, Basic, Ada) : ils vérifient de très près la conformité des instructions, d'abord à la traduction et par fois (Basic) à l'exécution en ajoutant les instructions nécessaires. Le Fortran est lui aussi très protecteur, mais on peut tromper sa vigilance grâce à sa très puissante instruction **EQUIVALENCE**. Les langages les plus permissifs sont le LM et le C. On peut les comparer à des pur-sang, capables d'emmener n'importe où qui saura les maîtriser. Les langages protecteurs seraient plutôt de confortables voitures roulant avec sûreté et aisance sur des routes bien tracées. On ne saurait trop conseiller au débutant de commencer par ceux-là.

---

(4) On rappelle qu'avec la plupart des systèmes d'exploitation, une fois que le programme a été *compilé*, puis *lié*, sa traduction en langage machine est placée dans un fichier (comportant le suffixe EXE sous DOS) qu'il suffit d'appeler pour le faire exécuter.

(5) Le Basic fourni avec le DOS avant DOS-5 n'en comporte pas, mais le QuickBasic ou d'autres Basic (comme PowerBasic...) disposent d'un *système intégré*.

(6) Lesquelles erreurs sont souvent appelées *bug* en anglais et *bogue* en franglais.

EXERCICES

Pour un lecteur néophyte, les exercices suivants seront sans doute encore difficiles à traduire en programme. Il pourra cependant en tracer l'organigramme, mettant en évidence spécialement les branchements et les boucles.

1 - Ecrivez un programme Basic qui recherche dans un texte (c'est-à-dire dans une chaîne), une par une, toutes les sous-chaînes *hon*, affiche les mots les contenant et attend une décision de l'opérateur pour :

– passer outre (par appui sur RC ou sur la barre espace),

– changer la sous-chaîne *hon* en *ho* (par appui sur la touche – ce qui signifiera enlever un *n*),

– changer *hon* en *honn* (par appui sur **+** : ajouter un *n*).

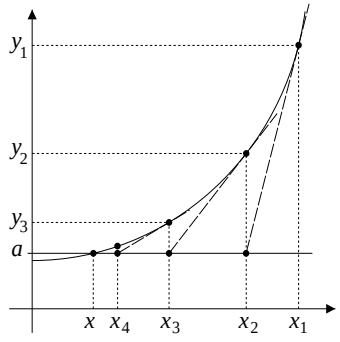
N'oubliez pas, en cas de modification, de décaler de ±1, selon le cas, l'indice de tous les caractères suivants. On rappelle que la fin d'un mot correspond à l'apparition du premier espace. Ce petit programme pourrait servir à un correcteur chargé de reprendre le texte d'un auteur ignorant les règles d'orthographe pour la famille *honneur* !

2 - Ecrivez un programme capable de trier une liste de nombres en désordre *nb(i)*. On pourra employer l'algorithme simple ci-après, bien que peu rapide. Pour tous les nombres *nb(i)*, à partir du deuxième, vérifiez si *nb(i)>nb(i*–1) . Si oui, passez outre. Si non, permutez ces deux nombres et donnez la valeur 1 à l'index *stab*. En fin de liste, si *stab* est nul, le tri est terminé. Sinon, remettez *stab* à 0 et recommencez à vérifier la position des nombres dans la liste.

3 - Ecrivez un programme imprimant les nombres premiers jusqu'à 1000 par exemple. Un nombre est dit *premier* s'il n'est divisible par aucun autre, sauf 1.

4 - Ecrivez un programme calculant la racine carrée x d'un nombre $y > 0$. On utilisera l'algorithme suivant. Partez d'une solution provisoire x par exemple égale à 1, ou à $y/2$ ou $y/4$ ... Calculez l'écart $d = y - x^2$. Si la valeur absolue de l'écart **abs**(d) est inférieure à la précision p qu'on se fixe, le calcul est terminé. Sinon, remplacez la valeur provisoire x de la racine par $x+(d/(2x))$ et recommencez.

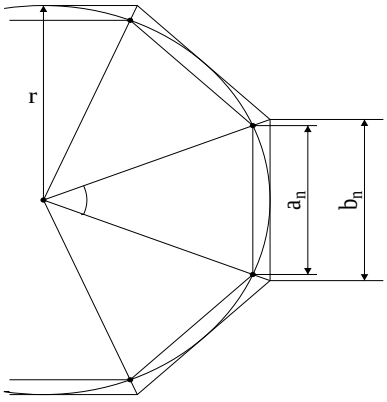
Cette méthode, appelée *méthode de Newton*, permet de résoudre beaucoup d'équations de type $y(x)=a$, où a est une constante, moyennant certaines conditions sur la fonction y (continuité, monotonie...). Elle repose sur les développements limités : en partant d'une première approximation x_n de la solution, on obtient une valeur meilleure en remplaçant x_n par $x_{n+1} = x_n + (a-y_n)/y'_n$, où y_n est la valeur de $y(x)$ calculée au point $x = x_n$ et y'_n sa dérivée au même point.



La figure ci-contre permet de voir comment la solution approchée progresse pas à pas vers la valeur exacte : le point de départ est x_1, à partir de quoi on calcule $y_1 = y(x_1)$, puis $y'(x_1)$, d'où la valeur calculée x_2 qui donne $y(x_2)$ et $y'(x_2)$, etc. La convergence vers la solution est assez rapide.

5 - Si on borde un cercle de rayon unité par deux polygones réguliers à n côtés (*n-polygones*), l'un inscrit, l'autre circonscrit, leur périmètre na_n et nb_n tend vers 2π en l'encadrant, quand n augmente indéfiniment. On écrira un programme permettant de calculer π comme limite des deux périmètres précédents. On doublera n à chaque pas. Les relations qui relient les côtés a_{2n} et b_{2n} des 2n-polygones aux côtés a_n et b_n des n-polygones sont les suivantes :

$$b_{2n} = a_n\, b_n / (a_n + b_n)$$
$$a_{2n} = \sqrt{a_n\, b_{2n} / 2}$$



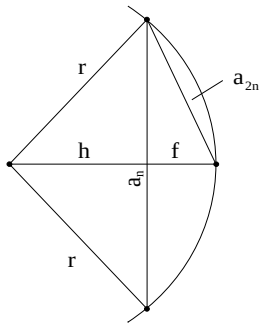
Commencer avec les carrés, c-à-d. avec n=4, $a_4=\sqrt{2}$ et b_4=2.

A partir de 5 décimales, il pourra y avoir des difficultés avec la précision. Il faudra probablement utiliser des réels de double précision. On rappelle que les 11 premiers chiffres de π sont donnés par le décompte des lettres dans chaque mot de la phrase mnémotechnique bien connue :

Que j'aime à faire connaître ce nombre utile aux sages ...!

6 - L'exercice précédent peut servir à montrer qu'il est possible de calculer le nombre π avec seulement des relations géométriques et un peu de patience (c'est le fameux problème de la quadrature du cercle !). Cependant les relations de récurrence employées sont difficiles à démontrer sans trigonométrie ; or celle-ci implique la connaissance préalable de π.

En ne s'autorisant que le théorème de Pythagore, on peut trouver une autre relation de récurrence. Pour le côté $a_{\mathbf{n}}$ du polygone inscrit, en se référant à la figure ci-contre, on obtient :



$$a^2_{\mathbf{2n}} = (r-h)^2 + a^2_{\mathbf{n}}/4$$

$$= 2r^2 - 2r\sqrt{r^2 - a^2_{\mathbf{n}}/4}$$

et, avec $r = 1$ et $c_{\mathbf{n}} = a^2_{\mathbf{n}}$,

$$c_{\mathbf{2n}} = 2 - \sqrt{4 - c_{\mathbf{n}}}$$

Cette relation, utilisée forcément avec des nombres de précision finie, conduit à des résultats catastrophiques par perte rapide de précision. Par exemple, au 12e pas, n = 16384, $a_{\mathbf{n}}$ = 0,000192..., $c_{\mathbf{n}} \approx 3{,}7\ 10^{-\mathbf{8}}$, l'expression $4-c_{\mathbf{n}}$ sous le radical, même avec les 16 chiffres significatifs des nombres en double précision, ne conservera au plus que 8 des chiffres de $c_{\mathbf{n}}$. Le radical est voisin de $2 - c_{\mathbf{n}}/4$, donc $c_{\mathbf{2n}}$ sera voisin de $c_{\mathbf{n}}/4$ avec guère plus de 7 chiffres significatifs !

Cette relation est donc sans intérêt pour des calculs précis. Cependant, on peut l'améliorer grâce à la formule bien connue $(x-y) = (x^2-y^2) / (x+y)$, qui conduit à :

$$c_{\mathbf{2n}} = c_{\mathbf{n}} / \left(2 + \sqrt{4 - c_{\mathbf{n}}} \right)$$

Cette fois-ci, la relation aboutira au bon résultat. En double précision, on trouve les 16 premiers chiffres de π en 25 pas environ.

Cet exercice avait pour but de montrer que, malgré l'énorme puissance des machines, le programmeur n'est pas dispensé de réfléchir et doit parfois trouver des *astuces* pour résoudre le problème posé. En tout cas, il fera bien de toujours chercher un moyen de vérifier la justesse de ses résultats. Par exemple ici, il faudrait effectuer le même travail sur le polygone extérieur, afin de vérifier que les deux périmètres tendent vers une limite commune. C'est l'exercice auquel nous invitons le lecteur intéressé.

Chapitre XI

LOGICIELS DE TRAITEMENT DE TEXTE

La grande majorité des calculateurs répandus de par le monde sont beaucoup plus utilisés comme auxiliaire de bureau que pour résoudre des calculs scientifiques. D'où le néologisme d'*ordinateur* qui, imposé par l'Administration, a connu un grand succès. Les logiciels alors employés sont regroupés sous le vocable de *bureautique*. La configuration matérielle est peu diversifiée : un clavier avec le maximum de touches (le pavé numérique est préférable si on emploie des tableurs), un écran agréable à lire, si possible en couleurs, mais sans performances spéciales, une imprimante de qualité, souvent *à laser* et partagée entre plusieurs postes de travail. Le disque dur est quasiment indispensable, ainsi que la souris.

Quatre grandes familles de logiciels dominent la bureautique : les *traitements de texte*, les *tableurs*, les *systèmes de gestion de base de données* (SGBD) et les logiciels *graphiques*. Nous consacrerons une étude à chacun d'eux. Bien que comportant beaucoup de différences de l'un à l'autre, ces logiciels obéissent à une logique suffisamment semblable pour qu'on puisse en dégager les traits essentiels.

Parfois ces logiciels sont groupés en un seul, appelé **logiciel intégré**. L'avantage de cette solution réside dans la faculté d'échanger des parties de documents entre ces logiciels. Par exemple, il est très intéressant de pouvoir inclure – on dit *importer* – un tableau, un dessin dans un texte, un rapport ... Ce n'est pas toujours facile, sauf dans deux cas : les différents documents sont fabriqués par des applications intégrées dans le même logiciel ou bien le système d'exploitation (Mac, Windows) assure lui-même ces échanges.

1 - PRESENTATION GENERALE.

Tous les grands logiciels de bureautique assistent l'opérateur en présentant à l'écran des **menus**. Ceux-ci affichent à un moment donné les choix possibles, dont certains d'ailleurs consistent en l'ouverture d'un nouveau menu plus détaillé que le précédent. On parvient finalement à l'énoncé de la fonction recherchée. On *clique* alors sur son nom avec la souris, ou bien on la *valide* en frappant la touche RC (*retour chariot*, *Enter*) après l'avoir mise en surbrillance en y amenant le curseur par les touches flèches. Tout cela s'apprend très vite : on dit que la présentation est *conviviale*, car elle est claire. En cas de faux pas dans la hiérarchie des menus, on revient en arrière en appuyant sur la touche Echappement (ou Esc, Escape) **(1)**.

Ce chapitre sera consacré aux logiciels de **traitement de texte**, parfois appelés *texteurs* et souvent **TTX** (en anglais, *word processor*). Ils apportent une aide remarquable à la frappe des documents et ont détrôné les machines à écrire, tant leur confort et leurs possibilités sont d'un niveau supérieur. Avec eux, la frappe se fait *au kilomètre*, c.-à-d. sans souci ni des fins de ligne, ni des fins de page, notions indépendantes de la frappe et modifiables à tout moment.

Pour obtenir une présentation correcte du document, on définira à part – avant, pendant ou après la frappe – la largeur et la longueur de la page, ainsi que les marges du texte et les retraits. Une fois ces carctéristiques définies, la mise en page est automatique, quelles que soient les corrections apportées par la suite au texte lui-même. Quelques TTX permettent même de diviser les pages en colonnes, comme dans ce livre, ainsi que d'insérer des figures, des tableaux, des images à l'endroit choisi.

On pourra souvent demander la numérotation automatique des pages. Avec les meilleurs TTX, le numéro de la page peut être imprimé à l'endroit désiré, en haut, en bas, au centre, au bord de la page, en variant même selon sa parité (page de droite ou page de gauche). On pourra demander qu'un en-tête ou un pied de page soit automatiquement ajouté à toutes les pages d'un même document (un chapitre de livre par exemple) ou bien à certaines d'entre elles.

Il est maintenant classique pour un TTX d'offrir la justification. On discutera ce procédé dans les sections 6 et 7, car son résultat est beaucoup lié aux caractéristiques de l'imprimante et de l'écran.

Tous ces *embellissements* sont spécifiés par l'insertion **automatique** dans le texte (ou à côté) de caractères spéciaux (caractères de commande), jamais imprimés et, pour la plupart, invisibles à l'écran.

---

(1) Le rôle ainsi dévolu à la touche Esc est très général en informatique, quel que soit le logiciel, par convention – sans doute tacite – entre développeurs.

2 - MODIFICATIONS

Une fois frappé, le texte est d'abord rangé en mémoire vive, puis en mémoire de masse, automatiquement ou à la diligence de l'opérateur. Même en mémoire de masse, il peut être aisément recopié dans la mémoire vive. Le délai de récupération variera beaucoup avec le type de mémoire dont il a fallu l'extraire ; c'est pour cela que le disque dur est fortement recommandé dès que le texte prend un peu d'ampleur.

Une partie du texte contenu en mémoire vive est présentée à l'écran, qui peut être considéré comme une *fenêtre* s'ouvrant sur le document (certains TTX autorisent l'ouverture de plusieurs fenêtres se partageant l'écran). Le texte affiché peut être corrigé, caractère par caractère ou par blocs. Pour supprimer, remplacer des *caractères* ou en insérer de nouveaux, on amène le curseur à l'endroit voulu. La touche *Suppr* (*Del*) supprime le caractère sous le curseur, tandis que la touche ← supprime celui qui le précède. Presque toujours, la touche *Inser* ou *Ins* permet de changer de *mode* de correction : *insertion* sans suppression au niveau du curseur, ou bien *remplacement* pur et simple du caractère en surbrillance.

Il est intéressant de savoir que le texte est sauvegardé dans des chaînes ou des tableaux de chaînes (une chaîne par paragraphe). L'insertion d'un caractère décale l'indice de tous les caractères situés derrière le nouveau. Tout cela est exécuté automatiquement et rapidement grâce aux instructions machine prévues pour la manipulation des chaînes.

La correction par *blocs* entiers de texte nécessite la *sélection préalable* du bloc (attention : le mot bloc n'a pas le même sens ici qu'en programmation), en marquant son début et sa fin à l'aide du curseur et de touches spéciales (par exemple en plaçant le curseur au début du bloc, puis, tout en maintenant MAJ enfoncée, en le plaçant en fin de bloc). Le bloc apparaît alors en *surbrillance* ou avec une couleur spéciale. Une fois sélectionné, ce bloc peut être supprimé *(coupé)*, recopié à un autre endroit (*collé, paste*), ou transféré (*coupé-collé*). La **souris** apporte une aide précieuse dans la manipulation du curseur et la sélection d'un bloc, car elle peut être déplacée dans le texte avec rapidité. On sélectionne un bloc de texte simplement en la traînant (*drag*) sur celui-ci, après avoir enfoncé son bouton (*cliqué*) au début du bloc.

En pratique, les modifications sont si faciles que la plupart des utilisateurs opèrent en deux étapes : ils commencent par frapper leur texte *au kilomètre*, sans souci de présentation ni de rigueur. Puis, dans un second temps, ils l'arrangent et l'amendent en fonction de la qualité recherchée.

3 - CARACTERES DISPONIBLES

Bien sûr, tous les caractères classiques des machines à écrire sont disponibles, y compris les lettres accentuées pourvu que le clavier et le logiciel aient été *francisé*s (avec, en ce qui concerne le clavier, si l'on travaille sous DOS, les instructions *ad hoc* dans le fichier **CONFIG.SYS**). Les lettres dotées d'accents graves ou aigus bénéficient d'une touche spéciale, tandis que celles requérant l'accent circonflexe ou le tréma se frappent en deux temps. Certains TTX sur les IBM-PC permettent d'utiliser les caractères du jeu étendu IBM en tapant leur numéro tout en maintenant enfoncée la touche ALT (voir page 44 ou l'annexe 1 pour se rappeler ces **caractères étendus**), mais les TTX disposent souvent de leur propre jeu dont il faudra connaître la liste en l'imprimant.

En imprimerie, les caractères revêtent des formes variées. Ils sont réunis en familles appelées **polices** *(fontes, fonts* en anglais), qui relèvent d'un même souci artistique : les lettres peuvent posséder un **empattement** *(serif)*, ce qui passe pour en faciliter la lecture, ou bien rechercher certains effets *(script)*. Ainsi la police *Garamond*, avec empattements, élégante mais sans fioritures, a pendant longtemps été considérée comme le summum dans l'art de la typographie.

Avec une machine à écrire, tout caractère occupe une largeur fixe, parce que son chariot avance de la même quantité à chaque frappe. Au contraire, dans l'imprimerie, la *matrice* de chaque caractère est moulée sur un petit rectangle de *plomb* de largeur variable. Par exemple, et c'est normal, un *m* ou un *w* occupera plus de place qu'un *i* ou un *l*. On appelle **polices fixes** celles dont tous les caractères possèdent une largeur fixe, et **polices proportionnelles** celles qui leur donnent une largeur propre. L'écriture avec une police fixe – celle des machines à écrire – est beaucoup moins esthétique que celle utilisant une police proportionnelle. De surcroît, l'écriture avec police proportionnelle est beaucoup plus condensée.

A l'instar d'une imprimerie, les meilleurs TTX disposent de plusieurs polices et permettent d'attribuer n'importe laquelle d'entre elles à tout caractère individuel, bloc, paragraphe, section, ... On peut de même bien souvent fixer la taille des caractères (leur *corps*) dans une large gamme indépendamment de leur police. Ces deux caractéristiques cependant sont liées aux possibilités de l'imprimante finale.

Enfin, certaines polices permettent d'écrire des caractères ne figurant pas dans la table ASCII, même étendue, comme l'ensemble de l'alphabet grec, des symboles mathématiques (tels ≤ ≥ ± ∞ ≠ → ⇔ ∃ ...) ou des caractères typographiques (™ ® ∴ ◊ ↵ • ‡ † ¶ ...). Il est fréquent qu'une telle police ait pour nom *symbol*.

4 - RENFORTS ET EFFETS SPECIAUX

Si seuls les meilleurs TTX disposent d'une large panoplie de polices à taille variable, tous, par contre, permettent le *renforcement* ou l'*habillage* des caractères, mots ou blocs par des attributs spéciaux, tels que *italique*, soulignement, **graissage**. On dispose parfois également du double soulignement, mais rarement du *surlignement*, pourtant bien utile dans les sciences, en particulier en logique (cf. chap. II).

Tous les TTX offrent la possibilité, pour les scientifiques, de placer des caractères en *exposant* ou en *indice*, plus rarement dans les deux positions simultanément sur un même caractère. L'effet de ces adjonctions est bien plus esthétique quand on en réduit la taille. Rien n'empêche d'utiliser le même procédé pour écrire l'indice ou l'exposant *devant* une lettre, comme l'exige le symbolisme des atomes en physique et en chimie nucléaires, mais rares sont les TTX autorisant indice et exposant simultanés **(2)** (ex.: $^{192}_{235}U$).

Citons pour mémoire des effets encore réservés à l'impression de qualité, comme la *ligature* (liaison entre deux lettres, comme f et l, f et i, o et e) ou le *crénage* (glissement d'une lettre sous la précédente pour éviter un espacement disgracieux, comme entre V ou T d'une part, r ou les voyelles d'autre part). Il est pourtant tout à fait possible que de tels raffinements soit bientôt disponibles dans les TTX de bureautique, comme ils le sont déjà en PAO *(publication assistée par ordinateur)*.

5 - TABULATIONS

A l'instar des machines à écrire, un TTX permet la tabulation, spécification de colonnes prédéfinies sur lesquelles se placera le curseur à la frappe de la touche ↔ (ou *Tab)*. Les marques de tabulation sont ou bien fixées une fois pour toutes par le logiciel ou bien – et c'est normal – ajustables par l'utilisateur pour chaque ligne ou chaque paragraphe. Sur ces "tabulations", il peut demander l'alignement du texte à gauche, mais aussi à droite ou demander son centrage.

Cette facilité permet la frappe de tableaux dans le texte même. Cependant, bien rares sont les TTX qui produisent sans peine des tableaux esthétiques, c.-à-d. avec des filets bien continus, du texte correctement placé dans les colonnes (centré, aligné à droite ou à gauche) et des caractères en rapport avec le reste du texte. Par contre, il leur arrive de pouvoir *importer* des tableaux plus présentables mis au point grâce à des logiciels spécifiques, les *tableurs* (chap. XII).

Les tabulations autorisent également le retrait de certaines lignes afin de les mettre en valeur. Cependant, la meilleure méthode pour mettre un paragraphe en retrait consiste à redéfinir ses marges, ce qui n'est pas à la portée de tous les TTX.

Pour insérer une illustration au droit d'un texte, il faut limiter la longueur d'un certain nombre de lignes (plutôt par redéfinition des marges d'un paragraphe). La partie restante en regard s'appelle *une réserve* ; c'est là qu'on importera l'image ou qu'on en collera une photocopie faite à la main.

6 - IMPRESSION

Les meilleurs TTX tirent parti de la majorité des possibilités des imprimantes, tout au moins de celles disponibles au moment de la création du logiciel. Celles commercialisées plus tard doivent être assimilables à l'une de celles reconnues par le logiciel. D'où l'apparition de **normes de fait** pour les imprimantes, qui, à moins de vouloir inaugurer un procédé révolutionnaire, se comportent comme – ou *émulent* – des imprimantes bien connues servant de référence.

En effet, le logiciel doit comporter un programme d'adaptation (*pilote* ou *driver)* traduisant **(3)** les codes de commande du TTX dans le langage de l'imprimante. Les premiers sont des caractères spéciaux, ceux dont le numéro ASCII est inférieur à 32 ou supérieur à 128. L'utilisateur appellera un menu (*initialisation*, *options imprimante*...) qui lui permettra de sélectionner le bon *pilote* en désignant son imprimante ou bien celle qui, dans la liste présentée, s'en approche le plus.

On peut distinguer deux types de langages d'imprimante : ceux qui impriment ligne par ligne (imprimantes matricielles, langages **HPLaserjet** ou PCL) et reposent sur des *séquences d'échappement*, chaînes de caractères commençant par *Esc* (ASCII n°27). Ils sont beaucoup plus simples que ceux de second type qui construisent en mémoire toute la page avant de l'imprimer. On appelle ces derniers *langages de description de page*. Le plus célèbre est PostScript, qui permet d'imprimer au bureau le plus complexe des documents, images comprises. Il est utilisé entre

(2) Lorsque rien n'est prévu, écrire indice ou exposant sur une ligne adjacente à demi-interligne et utiliser les tabulations pour aligner.
(3) Ils se réduisent parfois à un simple tableau de correspondance.

autres par les Macintosh pour les impressions laser. Windows avec GDI opère de même, mais de manière moins perfectionnée. Les langages de description de page exigent beaucoup de calcul et de mémoire (autant de bits que la page imprimée contient de pixels). Ces moyens sont fournis par l'imprimante dans le cas de PostScript et par Windows dans le cas de GDI.

L'imprimante matricielle ne peut imprimer que les caractères qu'elle connaît : elle doit en posséder le dessin. Si son langage est vectoriel (PostScript, GDI), il lui suffira de connaître les relations mathématiques qui dessinent les caractères dans un corps de référence – une police – unique ; tous les autres corps (les tailles) se déduiront de cette référence. Cette dernière doit être décrite dans les mémoires mortes fournies avec l'imprimante. Il est toutefois possible, mais laborieux, d'écrire de nouvelles polices dans le langage utilisé.

Si l'imprimante n'obéit pas à un langage vectoriel, elle doit posséder le dessin (le tableau de points) de chaque caractère dans un certain nombre de corps, qui seront seuls disponibles. Ces polices, avec des corps précis, peuvent être fournies à l'imprimante de trois façons différentes : dans ses mémoires mortes dès la conception de la machine ; par adjonction de cartouches (mémoires mortes) selon les besoins ; enfin, par téléchargement de polices logicielles depuis le calculateur.

7 - JUSTIFICATION ET CESURE

Tout TTX élaboré permet la *justification*, c.-à-d. l'alignement du texte à droite et à gauche. Pour y parvenir, les plus rudimentaires doublent l'espace entre certains mots (c'est peu esthétique). Les meilleurs assurent un espacement égal entre tous les mots d'une même ligne. Pour cela, ils comptent le nombre *N* de mots dans la ligne, calculent la longueur *L* de celle-ci, puis celle *l* occupée par les caractères (tous les caractères n'occupent pas la même longueur si la police est proportionnelle). Ils donnent alors à chaque espace une longueur égale au quotient *(L–l)/N*. Si cet espace s'avère un peu grand, le logiciel essaye de placer sur la ligne le premier mot de la ligne suivante. Le summum de l'art consiste à répartir ensuite entre chaque caractère l'espacement excédentaire, lorsqu'il est trop disgracieux : c'est la *justification fine*.

La présentation sera améliorée si le logiciel permet la **césure** (coupure de mots en fin de ligne). Tous ne connaissent pas cet artifice. Certains exigent une intervention manuelle à chaque césure. Les plus perfectionnés pratiquent la césure automatique, qui doit obéir à des règles très strictes pour obtenir un document de qualité. Il sera cependant toujours préférable que l'opérateur puisse corriger ou diriger la césure automatique. En résumé, tout bon TTX doit autoriser la césure, mais laisser la priorité à la césure manuelle, si l'opérateur en connaît bien les règles (elles ne sont pas les mêmes dans toutes les langues) **(4)**.

8 - L'ECRAN WYSIWYG

On se souvient que, lorsque l'écran est géré en mode caractère (cf. chapitre V, page 38), il est divisé en un certain nombre de cases (généralement 25×80) correspondant chacune à l'emplacement d'un caractère. Ni les polices proportionnelles, ni la justification fine ne peuvent alors être prises en compte. La seule justification admise est celle découlant de l'adjonction d'espaces supplémentaires. Un palliatif parfois utilisé consiste à attribuer à chaque ligne de texte un très grand nombre de caractères d'écran, dont seule une partie (la *fenêtre de vue*) sera visible à l'écran à un moment donné.

Cette présentation sera nettement différente de celle imprimée, surtout avec les polices proportionnelles de petite taille, qui permettent de placer bien plus de 80 caractères dans une ligne. Pour remédier à ce défaut, certains TTX autorisent une présentation en *mode aperçu (preview)* dont l'écran, cette fois-ci en mode graphique, donne une image assez exacte, mais peu lisible, de la page à imprimer. On ne pourra cependant exécuter aucune correction dans un tel mode.

Le *nec plus ultra* pour les TTX (ceux des Macintosh, ceux des IBM-PC sous Windows ...) consiste à employer l'écran exclusivement en mode graphique. La représentation est alors identique ou très voisine de ce qui sera imprimé : police et corps des caractères, renforts, longueur de ligne ... On parle alors de mode *wysiwyg* (abréviation de *what you see is what you get*, en français : *tel écran, tel écrit*). Ce procédé est presque parfait. Tout au plus doit-on remarquer que le jeu de polices de l'écran, plus réduit que celui de l'imprimante (pour ne pas encombrer la mémoire du PC), entraîne des substitutions de caractères généralement peu discernables. Ce mode gagne encore en confort avec un écran de taille supérieure aux 14" habituels (*écrans pleine page*).

(4) En français, on coupe en respectant l'étymologie et la construction, par exemple entre préfixe et racine, entre deux syllabes, entre deux consonnes, surtout s'il s'agit de consonnes doubles, mais pas entre deux consonnes *fondues* comme bl, fl, dr, ph, ch, gn. On ne coupe jamais après une apostrophe, ni entre un x, un y et une voyelle.

9 - GESTION DE FICHIERS

Le texte composé doit être sauvegardé dans un *fichier* approprié, et même sauvegardé fréquemment en cours de travail pour ne pas s'exposer à une perte trop grande en cas de panne. Parfois, avec l'autorisation de l'opérateur, le logiciel déclenche périodiquement la sauvegarde du document en cours.

Le nom du fichier de sauvegarde est généralement affecté d'un suffixe propre au logiciel utilisé, de façon qu'il sache reconnaître ses créations. En effet, le procédé normal pour reprendre le travail sur un texte existant consiste à mettre le TTX en exécution, puis à examiner le catalogue des fichiers reconnus par lui et à sélectionner celui recherché (à moins de se souvenir de son nom exact, auquel cas on peut toujours demander son ouverture d'emblée). Sur les Macintosh par contre, c'est le *fichier texte* qu'on sélectionne : cette sélection déclenchera l'appel du TTX qui l'a créé (voir chap. VIII, page 63, §4 et note 5 et pp. 68-69. Ce mode d'appel est également possible sous Windows.

Beaucoup de logiciels opèrent la sauvegarde en deux étapes. Ils rangent le texte à *sauver* dans le fichier normal (discuté ci-dessus) en *écrasant* le texte précédent. Mais auparavant, ce dernier a été automatiquement recopié dans un fichier de même nom, avec un suffixe différent, en général **.BAK**. Ainsi, une sauvegarde malencontreuse pourra se réparer, après retour sous le système d'exploitation, en renommant le fichier de secours *(back-up file)* avec un nom différent afin d'en récupérer le texte.

Certains TTX peuvent insérer dans le texte des fichiers extérieurs. Il faut cependant que ceux-ci soient compatibles avec le logiciel. Ceux créés par lui le sont toujours. Mais d'autres également, selon la puissance du TTX. En particulier, il est très souhaitable de pouvoir inclure (on dit *importer*) les produits des *tableurs* ou des graphiques. Ces facilités ne sont disponibles que dans les meilleurs TTX (ceux fonctionnant sous Macintosh ou sous Windows entre autres).

Il semble que cette fonction d'**importation** doive bientôt être considérée comme l'un des aspects importants d'un TTX. Le développement du rôle de l'**image** dans la communication entraîne la nécessité d'inclure schémas et illustrations dans les écrits. C'est d'autant plus souhaitable que, comme on le verra par la suite, il devient de plus en plus facile de créer des images informatiques. Les trois grands types de fichiers générateurs d'image qu'il nous paraît indispensable de pouvoir importer dans un texte sont ceux décrits sous forme de *points (bitmap)* ainsi que ceux écrits dans les langages vectoriels *HPGL* et *PostScript*.

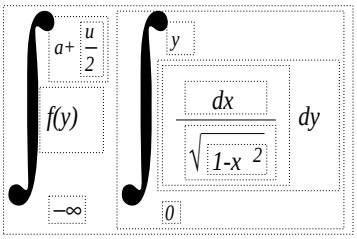
10 - FORMULES MATHEMATIQUES

Tout scientifique adepte de la bureautique recherchera un TTX sachant représenter correctement les formules mathématiques. Les concepteurs du jeu de caractères IBM *étendu* ont œuvré timidement dans ce sens en prévoyant quelques lettres grecques et des symboles comme :

$$\int \quad \sqrt{} \quad \pm \quad \leq \quad \geq \quad \equiv \quad \Leftrightarrow \quad \Rightarrow \quad \nabla \quad \wedge \quad \infty \quad \Sigma$$

Ces caractères sont loin de permettre l'écriture de formules agréables à lire. Peu de TTX offrent des services satisfaisants sur ce point. Le logiciel **T_EX** détient certainement la palme générale, mais on ne peut pas le classer dans les TTX, tellement il est peu convivial et d'emploi difficile (il s'agit plutôt d'un logiciel de *composition typographique* plus ou moins accessible aux PC, spécialement sous Unix). **Mathor** (marque de Novédit), permet également de bien représenter formules mathématiques et formules chimiques développées, mais, peu performant en ce qui concerne le texte ordinaire, il n'est plus guère employé. Les Macintosh permettent l'importation de formules créées avec facilité sous le logiciel **Expressionist**, d'une qualité tout à fait acceptable. Sous Windows, certains TTX permettent également l'écriture de formules mathématiques, mais cette possibilité est assez récente et souvent imparfaite.

Il peut être intéressant de savoir comment fonctionne un bon générateur de formules. Chaque "fonction" comme l'intégrale, la racine, la fraction, ... est placée dans une *boîte* élastique agrémentée d'un ornement (son symbole) et comportant des *sous-boîtes* : par exemple, trois sous-boîtes pour l'intégrale (deux pour ses bornes, l'autre pour l'intégrant) ... Ces boîtes sont encastrées les unes dans les autres, telles des poupées-gigognes. Leurs dimensions se modifient à mesure qu'on y écrit (ou qu'on en retire) des caractères, entraînant celles de toutes les boîtes plus extérieures. Bien que les limites de ces boîtes soient invisibles à l'impression, elles n'en sont pas moins réelles. En particulier, il faudra apprendre à entrer dans une boîte, en forçant sa frontière et celle des boîtes la contenant (grâce à la souris ou à des touches combinées avec *Alt* ou *Ctrl*).



11 - RAFFINEMENTS

Il est regrettable, on l'a vu, que certains TTX n'assurent pas des fonctions importantes et, à notre avis, élémentaires comme :

- l'écriture de n'importe quelle formule scientifique,
- l'utilisation des caractères semi-graphiques d'IBM,
- l'importation des fichiers de dessin les plus courants.

Cependant beaucoup d'entre eux proposent à l'écrivain des services supplémentaires, parfois d'un secours appréciable.

Le plus classique consiste en la **vérification orthographique** par un *dictionnaire*. On peut soumettre après frappe son texte à une analyse de ce type plus ou moins automatique. Pour l'instant du moins, aucun d'entre eux n'est fiable dans la vérification de la désinence des mots sous l'effet des règles grammaticales d'accord ou de conjugaison. Ils sont par contre très utiles en ce qui concerne les erreurs de vocabulaire dans le corps des mots, en particulier celles sur les lettres doubles ou les voyelles accentuées. Ils détectent de plus une grande partie des fautes de frappe les plus grossières. Par contre, il est courant qu'ils trouvent des fautes là où il n'y en a pas ; pour y remédier, il faut – si c'est possible – se constituer un dictionnaire personnel qui regroupera les mots les plus courants utilisés par l'opérateur et, à l'origine, inconnus du TTX.

On peut également trouver des dictionnaires de **synonymes** qui pourront être utiles aux auteurs en manque de vocabulaire. Ils doivent être employés avec beaucoup de circonspection, car les informations procurées sont souvent aberrantes. Cette possibilité, en général facultative, entraîne un encombrement supplémentaire important de la mémoire par ledit dictionnaire.

Le **publipostage** *(mailing)* est une facilité assurée également par beaucoup de TTX. Ce procédé permet d'envoyer à un grand nombre de destinataires une lettre qui va leur apparaître personnelle, mais qui en réalité comporte une majeure partie d'éléments banalisés (i.e. communs). L'auteur écrit une *lettre de base*, valable pour tous, mais dans laquelle il place des marques de *champs variables* là où devront être insérées des informations personnelles, telles que le nom, l'adresse du destinataire, ou tout autre élément lié à une personne ou à une société. Dans un autre fichier, l'auteur écrit les *données*, c.-à-d. l'ensemble des informations personnelles que le TTX devra substituer aux champs. Ce fichier comporte autant d'enregistrements ou *lignes* que la lettre en question a de destinataires et chacune de ses lignes contient autant de données (texte ou nombres) que la lettre contient de *champs*. Lors de l'impression de la lettre, à la rencontre d'un champ, le TTX vient chercher dans le fichier de lecture la donnée à substituer et écrit autant de lettres que ce fichier contient de lignes. On retrouve là encore des moyens informatiques déjà rencontrés en programmation. Certains TTX permettent même de faire dépendre l'impression d'une partie du texte de la valeur prise par un champ.

Dans un autre ordre d'idées, on relèvera la possibilité parfois offerte de créer une liste de **styles**. En effet, la grande diversité des attributs typographiques disponibles dans les TTX risque d'entraîner un manque d'homogénéité regrettable dans la présentation d'un gros document. On évitera cet écueil en définissant à part – dans une *feuille de styles* – des styles peu nombreux, complets (police, corps, retraits, marges, renfort ...) étiquetés par un ou deux symboles et applicables à n'importe quel bloc du document. A noter que les TTX assurent presque tous la *continuité* du style : si l'on insère du texte dans une ligne (par frappe directe), le style appliqué est automatiquement celui du caractère sur lequel on a placé le curseur d'insertion.

Certains TTX permettent des manipulations encore plus personnelles, comme la modification de la **table des caractères** disponibles (la police). En particulier, un écrivain français peut souhaiter imprimer le caractère œ au lieu de oe. De telles opérations sont toutefois difficiles : on expliquera dans l'un des exercices page 102 comment procéder pour un traitement de texte particulier.

Mentionnons en terminant le procédé appelé *secours à la veuve et l'orphelin*, qui consiste à ne pas tolérer en haut ou en bas de page une ligne seule, inesthétique, isolée du reste du paragraphe. Sur demande, le TTX peut gérer ce problème lui-même.

Ce sont là des raffinements qui témoignent de l'évolution progressive des TTX vers des logiciels de composition typographique, permettant à l'amateur de rivaliser avec un imprimeur professionnel, du moins s'il est prêt à consacrer à l'*art du bien écrire* tout le soin dont sait faire preuve cette corporation.

12 - PREAO

On ne confondra pas la PréAO avec la PAO, déjà citée (publication assistée par ordinateur) et qui, trop spécialisée, ne sera pas étudiée ici. PréAO veut dire *présentation assistée par ordinateur*. Avec un logiciel de ce genre, on compose des pages sur un écran d'ordinateur avec texte et illustrations, pages destinées à être projetées devant un auditoire au cours d'un exposé sur un écran mural de cinéma. Jusqu'ici, pour accompagner ce genre de présentation, on employait surtout les *transparents*, souvent écrits à la main avec des crayons feutre spéciaux et projetés sur le mur par un *épidiascope*.

Le logiciel de PréAO ressemble à un TTX, mais il possède des moyens d'illustrations spécifiques qui le rapprochent de la PAO. A l'aide de menus conviviaux, on peut remplir le fond de page avec des couleurs ou des motifs spéciaux, écrire du texte par dessus avec un grand choix de polices et de couleurs. On y insère des cadres ou des bordures diverses et il est facile d'y dessiner des organigrammes. On peut importer dans ces pages ou dans ces cadres des blocs de texte créés par un autre logiciel ou bien des images informatiques de tout genre ou presque. Les gestionnaires apprécient de pouvoir y inclure les graphiques des tableurs (cf. chapitre suivant). Les scientifiques préfèrent pouvoir y importer des schémas PostScript, possibilité déjà moins courante. Il est de plus souhaitable que ces logiciels produisent eux aussi des fichiers PostScript pour pouvoir les affiner ou les ajuster "à la main" avec toute la puissance qui caractérise ce langage.

Des *maquettes* ou des *vues maître*, peut-être un peu longues à mettre au point, réutilisables pour chaque vue, permettent de donner de l'homogénéité à la présentation. De petits dessins tout prêts (*cliparts*) peuvent être ajoutés ici ou là, bien que leur emploi de manière exagérée ne soit pas le meilleur moyen pour captiver un auditoire. Rappelons que, dans ce genre de présentation, il faut écrire avec des caractères de taille suffisante (corps 16 à 20) et limiter le nombre de lignes par vue à une valeur raisonnable (12 à 18).

On compose son travail de présentation vue par vue et on examine le résultat à l'écran de l'ordinateur. Ce n'est pas très difficile. C'est la suite qui l'est. Pour projeter en définitive ces images sur le mur, il faut un moyen (un périphérique?) supplémentaire. Le plus accessible est l'appareil de photo avec lequel on *prendra* chaque écran sur une diapositive 24×36. Le temps de pose sera long (donc commande par flexible de l'appareil sur pied). Ce temps dépend de la brillance de l'écran. On fera des essais en partant du temps moyen donné par la formule ci-après : $T = 25\ N^2/S$ où T est en secondes, S la sensibilité ISO (ou ASA), N est l'*ouverture* (1,8-2,8-4-5,6-8-11-16). Il est cependant difficile par ce procédé de reproduire exactement les couleurs de l'écran.

On peut également imprimer le fichier produit avec une imprimante couleur sur transparent. Les couleurs seront franches et la résolution excellente C'est un moyen déjà moins courant **(5)**. L'outil idéal est plus rare encore, c'est le projecteur informatique (voir chap. V, §3d). Qu'il soit à cristaux liquides ou de type cathodique (beaucoup plus lumineux, mais bien plus cher), il est capable de projeter en direct le dessin sur un mur ou un écran mural. C'est seulement avec ces appareils qu'on pourra exploiter facilement certaines fonctions avancées des logiciels de PréAO, à savoir l'animation automatique des images, ainsi que leur accompagnement sonore ou musical.

(5) Il existe des périphériques *(imageurs)* spécialement conçus pour produire des diapositives à partir de fichiers images d'écran. Des sociétés de service, dotées de ces appareils, assurent cette transformation par courrier.

EXERCICES

Le meilleur exercice pour apprendre à se servir d'un TTX consiste à se mettre au travail avec un logiciel donné. On pourra trouver ci-dessous deux "exercices" plutôt difficiles, à considérer plutôt comme des compléments de cours.

1 - Ecriture de formules mathématiques.

Si on ne possède pas de logiciel *ad hoc*, mais s'il est suffisamment souple, on peut procéder ainsi :

On emploiera pour l'intégrale, la somme ou le produit multiple... les caractères de la police Symbol (∫ Σ Π) et on décomposera la formule en un certain nombre de lignes (ou de paragraphes monolignes), dans lesquelles tous les caractères ont la même cote. On placera sur la ligne supérieure les bornes d'intégrale ou de somme ainsi que les exposants éventuels. De même pour les indices et les bornes inférieures, tous sur la ligne la plus basse. L'interligne sera réduit à 60 ou 70% de sa valeur normale. On assurera la correspondance des positions horizontales par des tabulations. Il en ira de même avec le symbole de racine carrée (caractère IBM 251 ou son équivalent prolongé par un trait – suite de tirets cadratins – sur la ligne supérieure écartée de 0,6 interligne). On peut écrire ainsi, avec des corps bien choisis :

$$I(a,b) = \int_a^b f(x)\,dx = \lim_{\Delta x \to 0} \sum_{i=0}^{(b-a-\Delta x)/\Delta x} f(a+i\Delta x)\,\Delta x \qquad x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$$

2 - Insertion de caractères spéciaux.

On dispose souvent d'une police dont certains caractères nous sont inutiles (caractères étrangers), alors que d'autres nous font défaut. Il est parfois possible alors de modifier les tables d'équivalence fournies avec le logiciel.

Par exemple, dans Word5 pour DOS avec imprimante PostScript, on peut modifier le fichier **POSTSCRA.IN5** ainsi : dans le paragraphe commençant par **/foreignvect**, on remplace le texte **/.notdef/.notdef** situé après **/questiondown** par **/OE/oe** . Si ce fichier pilote est bien celui utilisé pour imprimer (il faut le sélectionner dans *Sortie-Options-Imprimante*), la frappe de Alt 169 et Alt 170 fera imprimer respectivement Œ et œ.

Ce remplacement n'est pas suffisant si l'on veut écrire des paragraphes justifiés, car Word ignore l'encombrement de nos deux nouveaux caractères. Pour que la justification des lignes contenant Œ et œ soit correcte, il faut faire une modification plus délicate. On appellera le programme **TRANSDI5** et on lui demandera de nous traduire le fichier **POSTSCRA.DI5** en un fichier-texte appelé par exemple **TOTO** (option **T**). **POSTSCRA.DI5** contient les *tables de chasse* (les largeurs) des polices utilisées par **POSTSCRA.IN5**. On cherche les codes W*x* correspondants aux polices qu'on veut modifier. Par exemple, on trouve que la Times-Roman (police **F24**) utilise les tables **W15**, **W16**, **W17** et **W18** (normal, italique, gras, italique gras). Dans ces quatre tables, il faudra modifier la largeur portée derrière les indices 169 et 170 de façon à atteindre le but recherché. Attention : pour que la modification soit opérationnelle, il faut à nouveau appeler **TRANSDI5** et, avec l'option **P**, et lui demander de nous retraduire **TOTO** modifié en un **POSTSCRA.DI5** qui deviendra opérationnel.

Autre exemple avec Word5 et **POSTSCRA.IN5** : on obtient des caractères **surlignés** (nécessaires en logique) en choisissant le type *barré* dans *Format-Caractère*, si on a pris soin de remplacer la ligne définissant en PostScript le *barrement* (elle débute par **/LS** et finit par **bind def**) par la suivante (valable seulement pour le corps 10, pour les autres corps modifier le nombre 6 avant **add**) :

```
/LS { currentpoint /uy exch 6 add def /ux exch def 4 ssm } bind def
```

Chapitre XII

TABLEURS ET SYSTEMES DE GESTION BASES DE DONNEES

A - LES TABLEURS

Les logiciels appelés tableurs servent à composer des *tableaux*, représentations d'un ensemble de données dans un ensemble de **cases** formées par l'intersection de **lignes** et de **colonnes**. Mais il y a tableaux et tableaux ! Tout bon logiciel de traitement de texte (TTX) permet certes de composer des tableaux comportant des **rangées** (lignes ou colonnes) séparées par des **filets** (traits verticaux ou horizontaux) ; ces tableaux sont en quelque sorte figés, statiques : leurs données restent telles qu'on les a frappées.

Les tableaux créés par les *tableurs* sont d'un tout autre acabit : en effet leurs données peuvent **dépendre** les unes des autres par des **relations mathématiques**. Leur calcul sera automatique. Si on modifie l'une des valeurs du tableau, toutes celles qui en dépendent seront recalculées. On dit qu'on a affaire à des **tableaux dynamiques.**

Comme dans les TTX, des *menus* guident l'utilisateur dans l'emploi du logiciel. Ils sont presque toujours très explicites et on apprend vite leur rôle. S'ils sont absents de l'écran, il faut savoir comment les appeler : *clic* sur la souris ou frappe d'une touche spéciale (la barre / par exemple) sont les moyens usuels pour faire apparaître le menu principal, d'où, de proche en proche, découleront tous les autres. En cas d'erreur de *bifurcation*, on revient au menu précédent en appuyant sur la touche ESC (*échappement*), de la même façon que dans les TTX (note 1, page 95).

1 - RANGEES, LIGNES, COLONNES

Le nombre de **colonnes** (verticales) autorisé dans ces tableaux est assez grand (au moins 26, souvent 256) ; il dépasse nettement la capacité d'affichage horizontale de l'écran. A l'aide de la souris ou des touches-curseur, il faudra déplacer le tableau par rapport à l'écran (on dit qu'on déplace une *fenêtre de visualisation* ou de *vue* dans le tableau), pour permettre à l'écran d'en représenter la fraction étudiée.

On peut donner à chaque colonne la **largeur** souhaitée (mesurée en nombre de caractères ou en millimètres). Les colonnes sont repérées de deux façons : soit par un nombre, soit par une ou deux lettres (A...Z, puis, si leur nombre dépasse 26 : AA, AB, ...). Les deux modes peuvent coexister dans un même tableur.

Les **lignes** *(rows)* sont horizontales. Elles peuvent être encore plus nombreuses que les colonnes (parfois des milliers), mais souvent limitées chacune à une seule ligne de caractères (contrairement aux colonnes, leur taille est souvent invariable). On les repère presque toujours par un numéro. On appellera **rangée** indifféremment une ligne ou une colonne.

2 - CASES, ADRESSES ABSOLUES ET RELATIVES

Une **case** *(cellule, cell)*, est l'espace délimité par l'intersection d'une ligne et d'une colonne. Elle sera repérée par ses **coordonnées**, étiquettes des deux *rangées* composantes. Ainsi, la première case du tableau, celle du haut à gauche, est désignée par **A1**, (comme dans le jeu de bataille navale) ou par **L1C1** (Ligne 1, Colonne 1). Nous emploierons la notation générale suivante : l'**adresse** de la case située sur la *n*ième ligne et la *m*ième colonne sera représentée dans ce livre soit par **LnCm**, soit par **Xn**, *n* et *m* étant des nombres, **X** la *m*ième lettre de l'alphabet, tandis que **L** et **C** sont des initiales imposées.

Mais attention ! Cette double possibilité d'adressage des cases est encore compliquée par l'alternative *adresse absolue – adresse relative*. Cette caractéristique étant fondamentale dans l'exploitation d'un tableur, il est nécessaire de bien la comprendre.

> Une **adresse relative** est une adresse dont la valeur exacte **dépend de la case où elle est écrite.**

Dans le mode **LnCm**, une telle écriture désigne toujours une *adresse absolue*, c.-à-d. invariable, indépendante de l'endroit où elle est placée. Ainsi **L1C2** désigne toujours la deuxième case de la première ligne. Pour indiquer une *adresse relative*, on utilise la notation **L(±m)C(±n)**, où le symbole ± représente soit le signe +, soit le signe −. La case désignée par cette adresse dépend de celle où elle est écrite. Ainsi, le symbole **L(+2)C(−1)** écrit dans la case L10C8 désigne en réalité la case **L12C7** (10+2 → 12 et 8−1 → 7).

Dans le **mode Xn**, par contre, l'écriture toute simple *Xn* désigne une *adresse relative*. On verra sa signification réelle dans la section consacrée à la copie transposée. Pour désigner dans ce mode une *adresse absolue*, il faut faire précéder la ou les coordonnées *X* ou *n* par un symbole spécial, souvent le signe dollar. Ainsi **$A$1** désigne de manière invariable, tout comme **L1C1**, la première case du tableau, tandis que dans **A$1**, **A** est une adresse relative en ce qui concerne la colonne et **$1** désigne de manière absolue et invariable la ligne 1.

L'emploi de ces adresses n'a d'intérêt pratique qu'avec les formules. La distinction entre adresses relative et absolue n'a, quant à elle, d'importance que dans l'emploi de la *fonction copie*. Mais ce cas unique est très important. On le détaillera dans la section 7.

On peut voir ci-dessous le dessin partiel d'un tableau montrant en grisé les cases d'adresse absolue **C6** et **H10**, à écrire plutôt sous la forme **$C$6** et **$H$10**. En mode **LnCm**, ces cases s'appelleraient **L6C3** et **L10C8**, tandis que, toujours dans ce mode, une référence à **L10C8** placée en **L6C3** s'écrira **L(+4)C(+5)** .

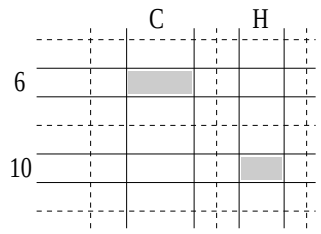


Fig. 12-1 : Coordonnées de cases

A tout instant, il existe une case privilégiée appelée **case active** ou **case de travail**. C'est celle dans laquelle se trouve le curseur si elle a été validée. Le curseur est déplacé de case en case par les touches flèches du clavier, ou avec la souris (cela dépend du tableur). On *valide* la sélection soit en frappant la touche **RC** (entrée), soit en *cliquant* la souris. La case active apparaît en surbrillance, ou avec une couleur spéciale. On ne peut écrire de modification que dans la case active. En outre, son adresse absolue et son contenu exact sont affichés dans une ligne spéciale, la **ligne d'état**, placée en haut ou en bas de l'écran (cette ligne d'état ne fait pas partie du tableau).

3 - CONTENU ET FORMAT DES CASES

Une case peut contenir une donnée numérique, du texte, une date, une formule ou même un autre tableau. On *peut* − et souvent, on *doit* − imposer le **format** de présentation de ladite donnée, comme lors des écritures produites par programmation. Par exemple, le *format* **F3** provoquera l'affichage du nombre contenu en virgule fixe avec trois chiffres après la virgule. Pour les dates, le format **JJ-MMM-AA** , s'il est autorisé, fera exprimer la date contenue dans la case par deux chiffres pour le jour, trois lettres pour le mois et deux chiffres pour l'année, chacun des groupes étant séparés par des tirets **(1)**.

Il faut se souvenir qu'il peut y avoir une différence notable entre le *contenu interne* d'une case et sa *présentation* à l'écran, qui est fixée elle **par le format**. Cette distinction a déjà été signalée à propos de la "sortie" programmée de variables (voir chap. X, §9, p. 91). Cependant, les tableurs étant conçus pour des non-informaticiens, une erreur dans le format ne conduira qu'à une perte de précision à l'affichage, au pire au rejet du format, alors qu'en programmation, une telle erreur peut provoquer un affichage erroné **(2)**.

Une case peut également contenir du texte en caractères ASCII (certains tableurs autorisent même l'emploi de polices proportionnelles). La plupart du temps, le texte doit débuter par un caractère particulier, l'apostrophe par exemple. Si ce texte déborde la largeur de la case, il sera mémorisé correctement, mais pas toujours entièrement représenté à l'écran. Il ne pourra l'être qu'en occupant la place dévolue aux cases voisines, ce qui sera éventuellement possible si elles sont vides.

Bizarrement peut-être, l'un des formats les plus utiles est le **format caché** ou *hide*. Comme son nom l'indique, il interdit affichage et impression du contenu de la case en question. On pourra placer dans celle-ci un résultat de calcul intermédiaire sans intérêt pour le destinataire du tableau final. Ce format permet donc une meilleure présentation du tableau.

On peut ajouter à un tableau de nouvelles rangées (vierges) par la fonction **insertion**. Hélas, on perturbe souvent ainsi un tableau déjà bien construit, surtout s'il contient des formules avec adresses relatives ; sa reconstruction peut devenir inévitable.

(1) La représentation *interne* d'une date est un entier, égal au nombre de jours écoulés depuis une date arbitraire (le 1er janvier 1980 par exemple). Ce procédé facilite le calcul des durées (soustraction) et des échéances (addition).

(2) Si la case du tableau est trop petite pour représenter un **nombre** dans le format fixé, celui-ci n'est pas tronqué, mais une marque d'erreur apparaît dans la case.

4 - FORMULES ET VARIABLES

La puissance des tableurs réside dans leur aptitude à recevoir des formules mathématiques dans leurs cases. Pour ne pas être confondues avec du texte, ces formules **doivent commencer par une marque particulière** [3]. Peuvent figurer dans une formule :

– les *opérateurs arithmétiques* usuels, ainsi que le **IF** ;

– des noms de *fonctions mathématiques* (parfois caractérisées elles aussi par un symbole spécial placé en tête, comme l'*arobasse* @) ;

– des *constantes*, c.-à-d. des valeurs numériques ou logiques, comme dans les langages de programmation ;

– des *variables*.

Une **variable** ne peut être que le contenu d'une autre case (ou d'un ensemble de cases) et se désigne par l'**adresse** de celle-ci (adresse relative ou absolue), ou quelquefois par un nom, à condition que l'on ait fait correspondre ce nom avec l'adresse de la case impliquée. Ainsi, on pourrait écrire dans la case **H7** la formule **=J5+B2** , si le symbole identifiant les formules est le signe = . Cette notation signifie évidemment que le contenu de **H7** sera égal à la somme du contenu de **J5** et de celui de **B2**. A l'écran ou lors d'une impression, sauf demande expresse, c'est le résultat de l'opération qui sera présenté avec le format fixé et non la formule réellement écrite (si on veut revoir quelle formule on a inscrite dans une case, il faut rendre cette case active et examiner la *ligne d'état*, cf. § 2, page précédente).

Donnons un autre exemple à l'emploi des formules. On pourrait écrire dans la case **H3** la formule

= $B$1 * @sin ($B$2 * F3 + $B$3)

ou bien : **= L1C2 * SIN (L2C2 * L(+0)C(-2) + L3C2)**

qui représentera la fonction : $y(t) = a \sin(2\pi t/T + \varphi)$

La valeur de l'amplitude *a* est placée en **B1** (ou **L1C2**), celle de $2\pi/T$, période angulaire, en **B2** (**L2C2**), celle de la phase φ en **B3** (**L3C2**) et celle du temps *t* en **F3**, case qu'on peut désigner en **H3** par **L(+0)C(–2)** puisqu'elle est située sur la même ligne **(L+0)**, deux colonnes plus à gauche **(C–2)**, que la case H3. L'emploi d'adresses tantôt absolues, tantôt relatives, n'est pas fantaisiste du tout si on désire faire varier le temps *t* (on en saisira tout l'intérêt dans la section 7).

5 - LA PRESENTATION DU TABLEAU

Certains tableurs délimitent automatiquement colonnes et lignes par des *filets* ou du *quadrillage*. On pourra supprimer en partie ou en totalité ces filets, rangée par rangée ou case par case, afin de donner une allure impeccable à la page imprimée, ne laissant subsister que des tableaux partiels ou même des cases isolées dûment mises en valeur par encadrement.

Tous les tableurs ne disposent pas de cette fonction. Si l'on tient au quadrillage, il faudra parfois le dessiner soi-même en lui réservant des lignes et des colonnes de largeur une unité. On y placera les caractères semi-graphiques adéquats [4] (cf. fig.6-2, page 44). La fonction *copie* de case à case, puis de ligne à ligne et enfin de bloc à bloc raccourcira beaucoup cette opération qui autrement serait fastidieuse.

Le tableau est donc représenté avec ou sans quadrillage, lequel est d'ailleurs localement masquable en lui attribuant ici ou là le format *caché*. Ce même format, on l'a vu, permet de cacher également le contenu de certaines cases ; tous ces formats, en règle générale, permettent une présentation fine et précise du contenu des cases.

L'impression d'un tableau donne un résultat proche de sa représentation à l'écran. Il n'y a évidemment pas impression des cases cachées. Une difficulté peut surgir si l'on a écrit soi-même les filets avec des caractères semi-graphiques et que l'imprimante ne les reconnaisse pas (elle les remplacera par d'autres, ce qui sera horrible). Ajoutons qu'il est possible de n'imprimer qu'une partie d'un tableau – une *plage* – qu'on aura au préalable sélectionnée comme il sera expliqué ci-après.

(3) Parmi les marques possibles, citons le signe égal, tout opérateur arithmétique, le nom de n'importe quelle fonction... Ces conventions dépendent du tableur utilisé.
(4) On rappelle qu'on les obtient sur les IBM-PC en maintenant la touche ALT enfoncée et en tapant leur numéro.

6 - SELECTION DE PLAGE

Il est possible de sélectionner tout un sous-ensemble du tableau, que nous appellerons *plage (range)*. Cette sélection est nécessaire avant d'effectuer une copie, ou pour affecter un même format à toute une plage ou pour en assurer l'impression.

Pour sélectionner une **rangée** entière, ligne ou colonne, on peut parfois *cliquer* sur son en-tête (son numéro). La **plage** sélectionnable la plus générale est un **rectangle**, dont toutes les cases font partie de la sélection. On appellera *coins* les cases situées aux angles de ce rectangle. La **sélection d'une plage** avec la **souris** s'obtient en l'amenant sur l'un des coins de la plage, en *cliquant*, puis en la *tirant (drag)*, bouton enfoncé, au travers de la plage jusqu'au coin opposé.

La **sélection par le curseur** est un peu plus difficile. On entre dans le menu approprié et on choisit les indications *ligne*, *colonne* ou *plage*. Pour définir une rangée, il suffit que le curseur sélectionne ensuite l'une des cases de la rangée. Pour une plage, on place le curseur sur l'un de ses *coins*, on frappe au clavier un *caractère de création de plage* (le *point*, les *deux-points* ..) et on amène le curseur sur le coin opposé.

La sélection est également possible (3e méthode) par la *ligne d'état*, en y inscrivant, quand les menus le demandent, les adresses des deux *cases de coins*, séparées par le symbole spécial de création de plage.

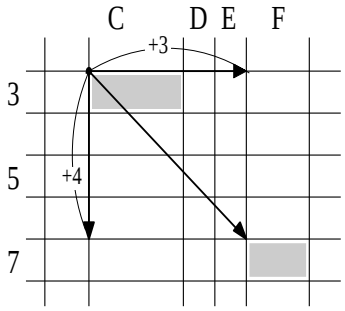
7 - LA COPIE DE PLAGE

La fonction **copie** permet de recopier le **contenu** d'une case, rangée ou plage dans une autre. Plus généralement, elle recopie une *plage émettrice* dans une *plage réceptrice*. C'est le contenu complet (format compris) qui est copié et non la représentation à l'écran. Si la plage émettrice ne comporte pas plus d'une rangée et si la réceptrice en comporte plusieurs, il y aura **répétition de la copie** dans ces dernières.

La copie ne pose guère de problèmes si la ou les cases copiées ne comportent pas de formules contenant des *adresses* (i.e. des variables de type **Xm** ou **LnCn**). Lorsque ce sera le cas, il faudra bien se souvenir de la **différence profonde** entre adresse relative et adresse absolue.

La notation de **type LnCm** ne soulève aucune difficulté. Adresses absolues **LnCm** ou relatives **L(±n)C(±m)** sont recopiées telles quelles. Les premières désigneront encore la même case, tandis que les adresses relatives désigneront à leur nouvel emplacement, une case dont la **position relative est la même que dans la configuration de départ**.

En notation **type Xn**, le résultat est exactement le même. Les **adresses absolues** (identifiées par un ou deux symboles $) sont recopiées telles quelles dans la ou les cases réceptrices. Mais les **adresses relatives** (cas habituel) sont, elles, **transposées**. Toute copie implique un *vecteur de translation*, dont les coordonnées sont égales au nombre de cases séparant dans chaque direction case émettrice et réceptrice. Ce vecteur est ajouté avant copie à toutes les adresses relatives de type **Xn**. Par exemple, si on copie la formule **=C2+B3** de la case **C3** à la case **F7**, on trouvera dans **F7** la formule **=F6+E7**, le vecteur de translation ayant pour coordonnées (3,4). En notation **LnCm**, en C3 on aurait écrit **=L(–1)C(0)+L(0)C(–1)**, ce que la copie n'aurait pas transposé, mais le résultat aurait été le même.



Ce procédé, extrêmement puissant, permet de programmer très simplement des opérations répétitives sur tout un ensemble de cases en ne les écrivant en réalité qu'une seule fois. On retrouve là l'équivalent de la notion de boucle étudiée dans les chapitres précédents : dans celle-ci, l'indice permet d'effectuer un même traitement sur tout un ensemble de variables (les *tableaux*) ; dans les *tableurs*, c'est le *binôme* **copie-adresse-relative** qui assure ce résultat. On étudiera avec soin l'exemple suivant ainsi que celui terminant cet exposé sur les tableurs.

Reprenant l'exemple du §4, on va étendre à tout un ensemble de valeurs de t la relation

$$y(t) = a \sin (2\pi t/T+\varphi)$$

avec a, T et φ donnés. Il y aura par exemple 21 valeurs de t échelonnées de 0 à T. On écrira la valeur de la période T en **B5** (60 par ex.), celle de l'incrément $\Delta t = T/20$ **(=B5/20)** en **B4** et, en **B2**, la période angulaire $2\pi/T$, c'est-à-dire **@PI∗2/B5** (on suppose que π est donnée par une fonction appelée **@PI**). Puis on écrit les valeurs 0 (t initial) en **F2**, puis la formule donnant le temps t, soit **=F2+B$4** en **F3** .

	B	D	F	H
1	100			
2	@PI*2/B5		0	0
3	1.571		=F2+B$4	=B$1*@SIN((B$2*F3+B$3)
4	=B5/20		=F3+B$4	=B$1*@SIN((B$2*F4+B$3)
5	60		=F4+B$4	=B$1*@SIN((B$2*F5+B$3)
6			=F5+B$4	=B$1*@SIN((B$2*F6+B$3)
...	.....		...	.....
21			=F20+B$4	=B$1*@SIN((B$2*F21+B$3
22			=F21+B$4	=B$1*@SIN((B$2*F22+B$3)

On écrit en **H2** la formule donnant y, à savoir **=B$1*@sin(B$2*F2+B$3)**, qu'on s'empresse de recopier en **H3** (où elle subit la transposition **F2** → **F3**).

Enfin, on procède à la *copie* de l'ensemble des deux cases **F3..H3** sur une plage formée de 2 colonnes et 19 lignes s'étendant de **F4** à **H22**. Avec la transposition des adresses relatives **F2** et **F3** et le maintien des adresses **B$1**,...**B$4**, contenant des paramètres fixes, on obtient bien une suite croissante de valeurs de *t* d'incrément Δ*t* dans la colonne **F** et la valeur *y(t)* dans la case de la colonne **H** en regard **(5)**.

Cette explication peut paraître bien longue. Quand on aura compris, on verra que la copie transposée est bien plus facile à réaliser qu'à expliquer, tant elle relève du bon sens.

8 - FONCTIONS ET CALCUL

De nombreuses **fonctions mathématiques** peuvent être incorporées dans les formules, de sorte que les tableaux ainsi créés peuvent être comparés à de véritables *programmes séquentiels* écrits à l'aide des langages évolués (chapitres IX et X). La mise au point de ce genre de programme est même beaucoup plus facile **(6)**, le calcul se déroulant en *temps réel* et passant par des étapes intermédiaires dont le résultat peut être rendu visible à l'écran.

Si l'on **change la valeur d'une donnée** dans une case du tableau, le logiciel – automatiquement ou sur demande – provoquera le **calcul** de toutes les cases qui contiennent une référence, directe ou indirecte, à la case modifiée. A cause de cela, ces tableaux sont qualifiés de *dynamiques*, puisqu'ils sont remis à jour à chaque altération.

Parmi les fonctions autorisées dans les tableurs, on retrouve une bonne partie des fonctions mathématiques rencontrées dans les bibliothèques des grands langages de programmation, telles que les fonctions *trigonométriques, exponentielle, logarithmes* ... On dispose bien sûr des possibilités de *branchement conditionnel* avec l'équivalent du **IF**. On y rencontre de plus des fonctions spécialement prévues pour les *calculs financiers* ou statistiques, manipulant les notions d'intérêt, simple ou composé, de sommes, de moyennes, d'écart-type ... Il va sans dire que les tableurs sont très employés dans les banques et les services financiers.

(5) Il n'est pas indispensable de placer le symbole $ devant une coordonnée absolue si elle ne doit jamais subir de translation sous l'effet d'une copie.
(6) Naturellement, leur temps d'exécution est beaucoup plus long.

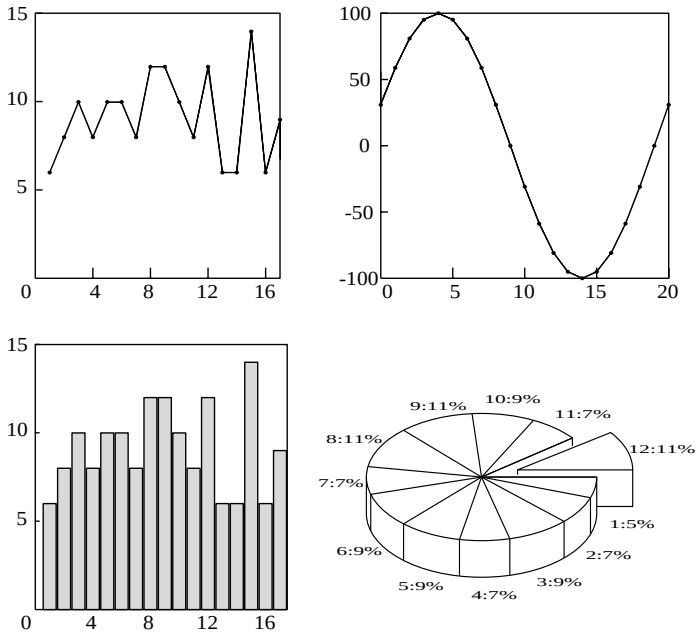
9 - GRAPHIQUE ET BASE DE DONNEES

La liste des atouts des tableurs ne s'arrête pas là. D'autres possibilités leur sont généralement adjointes, dont on ne discutera que très peu, car on empiéterait sur la matière des chapitres suivants.

On a vu qu'il était tout à fait possible avec un tableur de faire calculer une rangée de *n* cases comme une *application* (au sens mathématique du terme) d'une formule à un ensemble de valeurs figurant dans une autre rangée de *n* cases (ensemble de départ). Une nouvelle facilité du tableur va permettre de demander la **représentation graphique** de cette *application* après avoir sélectionné l'ensemble de départ, puis celui d'arrivée. Le graphique n'est peut-être pas aussi *fignolé* que celui que mettrait au point un programmeur chevronné, mais il est beaucoup plus facile à obtenir.

En matière de graphique, l'utilisateur dispose même d'un choix : tracé cartésien, tracé sous forme de barres, sous forme de secteurs (ironiquement appelée *camembert*). Les figures ci-contre montrent quatre de ces graphiques : celui du haut à droite représente la sinusoïde étudiée §7 (plage **F2..F22** en abscisse, plage **H2..H22** en ordonnée) ; les autres montrent les trois formes classiquement disponibles appliquées à un autre exemple de fonction, celle donnant le nombre de pages contenues dans les différents chapitres de ce livre (la représentation en *camembert* a été limitée aux douze premiers chapitres, pour ne pas surcharger la figure).

La plupart des tableurs acceptent également des fonctions de recherche qui leur permettent de travailler comme un SGBD. Ces fonctions s'appliquent aux contenus des cases et le tableau peut alors être regardé comme une **base de données** dont les *fiches* sont les lignes et les *champs* sont les cases. Sa capacité et sa puissance cependant sont loin d'atteindre celles des SGBD que nous allons étudier dans la deuxième partie de ce chapitre. Le lecteur qui aura bien compris leur fonctionnement n'aura pas trop de difficultés à en appliquer les principes aux opérations de tris et de recherches souhaitées dans le tableau.



Nombre de pages par chapitre dans ce livre : graphique cartésien en haut à gauche, en barres et en camembert en bas (12 chapitres seulement pour ce dernier). En haut à droite, sinusoïde étudiée dans les paragraphes précédents.

10 - EXEMPLE DE TABLEAU

Pour illustrer le fonctionnement d'un tableur, on va créer ensemble une feuille de calcul donnant le plan de remboursement d'un emprunt.

Le problème posé est le suivant : vous voulez emprunter auprès d'une banque mettons 10 000 F (ce montant n'a pas d'importance, toutes les sommes calculées lui seront proportionnelles). On vous propose diverses formules, variables en durée et en taux d'intérêt. Ce taux est toujours fixe durant toute la période de remboursement, le prêt est du type à intérêts composés et les échéances constantes. On va se donner un moyen d'étudier facilement toutes les propositions.

A chaque échéance (on admettra ici que leur périodicité est trimestrielle), vous rembourserez une somme fixe R qui comprend l'intérêt dû sur le capital restant, plus un remboursement partiel de celui-ci. Pour un capital C_0 emprunté, un taux d'intérêt τ par échéance (ici taux annuel divisé par 4), le premier versement R comprendra l'intérêt $C_0\tau$ sur le capital initial, plus une fraction de capital égale à $R - C_0\tau$, de sorte que le nouveau capital C_1 sera $C_0(1+\tau) - R$. Après paiement de la *ième* annuité, le capital restant sera donné par la relation de récurrence

$$C_i = C_{i-1}(1+\tau) - R\ ,$$

qu'un peu d'algèbre résout en

$$C_i = C_0(1+\tau)^i - R\sum_{k=0}^{k=i-1}(1+\tau)_k = (C_0 - R/\tau)(1+\tau)^i + R/\tau$$

En particulier, on calcule aisément le montant des N échéances R remboursant un capital initial C_0, en faisant $i = N$ et $C_N = 0$ dans la relation ci-dessus. On obtient la relation suivante :

$$\frac{\tau C_0}{R} = 1 - \frac{1}{(1+\tau)^N}$$

Par exemple, on rembourse 10 KF avec des échéances de 796,11 F tous les 3 mois sur 4 ans (N=16), si l'intérêt composé est de 12% l'an.

Ces calculs sont assez complexes. On va voir qu'un tableur permet de résoudre le même problème avec beaucoup plus de facilité, puisqu'il nous dispensera de tout calcul algébrique. Le tableur utilisé ressemblera comme un frère à un logiciel certes peu perfectionné, mais en libre circulation (ASEASY). Les formules y sont introduites par les signes + ou – . Notre feuille de calcul comportera :

– tout d'abord 10 lignes réservées à des **titres** toujours souhaitables (on les inscrira dans les cases, même s'ils en débordent la largeur) ;

– en 12e position, une ligne donnant l'*intitulé* de chaque colonne et encadrée par 2 filets horizontaux (occupant, eux, les lignes 11 et 13) ;

– à partir de la 15e, des lignes disponibles pour la liste des mensualités, ou plutôt des trimestrialités de remboursement ;

– cinq colonnes utiles, A, C, E, G et I, d'une largeur de 12 caractères, sauf la première qui en aura 14, et intitulées *échéance*, *intérêt*, *capit. remb.*, *cap. restant* et *rembours.* ;

– cinq colonnes (B, D, F, H et J) de largeur une unité, qui contiendront les filets verticaux délimitant les colonnes utiles.

Si notre tableur ne dessine pas lui-même les *filets*, nous le ferons à sa place. On remplit toute la colonne B à partir de la ligne 11 (grâce à la *copie* d'une première case) avec des caractères *Alt*-179, sauf en **B11** (*Alt*-194) et **B13** (*Alt*-197), puis on copie la colonne B dans les colonnes D, F, H et J . On remplace les caractères en **J11** et **J13** par *Alt*-191 et *Alt*-180. On place 14 caractères *Alt*-196 en **A11**, qu'on recopie dans **A13**. On copie ensuite le bloc **A11..A13** en **C11..C13**, en **E11..E13** et en **I11..I13**. Le quadrillage ainsi obtenu est presque parfait.

Les *formats* sont attribués ainsi : **T** (texte) à la colonne A, aux lignes 3 et 12 et aux cases **A6**, **A8**, **G8** ; **F2** (fixe, deux décimales) aux colonnes C, E, G, I . On écrit ensuite dans la ligne 12 les titres convenant à chaque colonne utile (cf. figure ci-contre), puis, dans la colonne A, l'intitulé d'un nombre suffisant de trimestres à venir.

	A	C	E	G	I
1					
2		TABLEAU DE REMBOURSEMENT DE PRET			
3					
4					
5	Emprunt	10000			25 sep 94
6					
7	Remboursement	800		Intérêt	0.12
8					
9					
10					
11					
12	Echéance	Intérêt	Cap. remb.	Cap.rest	Rembours.
13					
14				10000.00	
15	janvier 95	300.00	500.00	9500.00	800.00
16	avril 95	285.00	515.00	8985.00	800.00
17	juillet 95	269.55	530.45	8454.55	800.00
18	octobre 95	253.64	546.36	7908.19	800.00
19	janvier 96	237.25	562.75	7345.43	800.00
20	avril 96	220.36	579.64	6765.80	800.00
21	juillet 96	202.97	597.03	6168.77	800.00
22	octobre 96	185.06	614.94	5553.83	800.00
23	janvier 96	166.61	633.39	4920.45	800.00
24	avril 96	147.61	652.39	4268.06	800.00
25	juillet 96	128.04	671.96	3596.10	800.00
26	octobre 96	107.88	692.12	2903.98	800.00
27	janvier 97	87.12	712.88	2191.10	800.00
28	avril 97	65.73	734.27	1456.84	800.00
29	juillet 97	43.71	756.29	700.54	800.00
30	octobre 97	21.02	700.54	0.00	721.56
31					
32	Récapitul.	2721.56	10000.00		12721.56
33					

En **C6**, on écrit la somme empruntée, en francs : 10 000 ; en **I8**, le taux annuel d'intérêt (ici 0.12), en **C8** la valeur du remboursement trimestriel fixe (par ex. 800). Ces valeurs sont un point de départ, on pourra ensuite les changer à sa guise afin d'adapter le plan aux possibilités de l'emprunteur dans les limites fixées par le prêteur.

On tapera ensuite les formules **+$C$6** en **G14** (capital initial à rembourser), **+G14*$I$8/4** en **C15**, **+I15-C15** en **E15**, **+G14-E15** en **G15** et **+C$8** en **I15** (on s'exercera à retrouver le bien-fondé de ces formules en se reportant à la page précédente). On recopie enfin l'ensemble des cases **C15..I15** dans la plage **C16..I***n*, le nombre *n* étant suffisamment grand (16) pour parvenir au remboursement total.

Le tableur est complètement programmé pour résoudre le problème posé et on verra apparaître toutes les valeurs souhaitées. On peut changer les valeurs initiales : le logiciel recalculera l'ensemble du tableau. On pourra ajuster R (**C8**) pour obtenir juste 0 en **G30**. R ne sera pas une valeur ronde (on trouve la même valeur qu'avec la formule algébrique : 796,1085). Si on préfère pour R la valeur ronde de 800 francs, il apparaîtra en G30 une valeur négative,

trop-perçu par la banque. Si on le désire, on peut programmer la dernière ligne (n°30) pour que ce trop-perçu vienne diminuer le montant de la dernière échéance ; c'est ce qu'on a fait dans l'exemple présenté en plaçant les formules **+G29** en **E30** et **+C30+E30** en **I30**.

On peut perfectionner le tableau en dessinant des cadres à double trait *(Alt*-201, 205..., 187, 186 ...) ou à simple trait autour des cases titres. Dans la case **I6** ainsi entourée, on pourra spécifier le format de date **JJ-MMM-AA** et on lui affectera la fonction **@date**, si elle existe dans le logiciel utilisé. Cette case recevra alors automatiquement la date du calculateur lors de chaque intervention dans le tableau.

On peut ajouter en bas du tableau une ligne de **récapitulation**, par exemple les formules **@SUM(C15..C30)** en **C32**, **@SUM(E15..E30)** en **E32** et **@SUM(I15..I30)** en **I32**. Grâce à elles, la case **C32** donnera l'intérêt **total** payé, **I32** le coût total, tandis que **E32** doit redonner la valeur du capital emprunté.

B - LES SYSTEMES DE GESTION DE BASES DE DONNEES

Ces logiciels gèrent des *données organisées*. Il est courant, dans les entreprises, les administrations... d'avoir recours à des *fichiers*. Ces fichiers peuvent concerner par exemple les clients, les articles produits, ceux en stock, les membres d'une association...(7) De telles informations ou *données* étaient autrefois écrites sur des *fiches* classées dans un fichier. Une fois informatisées, ces données continuent d'être appelées *fichier*, mais plutôt par les non-spécialistes, car ce terme désigne déjà un autre concept en informatique (ensemble d'informations sur un support sans organisation particulière, cf. chap. VI, §15, p. 49). Aussi, les informaticiens appellent **bases de données** les ensembles dont nous parlons.

Dans les *bases de données*, les fiches ont toutes une organisation similaire. Par contre *à l'intérieur d'une fiche*, on peut rencontrer des informations très diverses, des nombres certes, dont le numéro de la fiche, des dates, des variables logiques, mais surtout du texte et même des images. On reconnaîtra là l'objet *structure* évoqué dans les chapitres VI et X. Ce sont bien en effet des structures que manipulent les **systèmes de gestion de bases de données** (en abrégé, **SGBD**), mais l'utilisateur n'a pas à s'en soucier.

Les logiciels appelés SGDB permettent de :

- de **créer** une base de données,
- d'en **définir** la structure (les **champs**),
- d'**écrire** dans les différents champs de chaque fiche,
- de les **rappeler** dans n'importe quel ordre pour les modifier ou les consulter,
- et surtout d'effectuer des **recherches** rapides et puissantes sur ces fiches, avec des *critères de sélection* variés.

(7) Ces derniers fichiers sont soumis à la législation en vigueur (chap. XVII).

1 - MOYENS D'ACCES AUX DONNEES

Le SGBD s'interpose donc entre l'utilisateur et les données rangées dans la base comme un outil apte à leur manipulation. Le SGBD peut se présenter sous deux formes : celle d'un langage de commandes ou une forme graphique. On retrouve là les deux aspects que revêtent les systèmes d'exploitation : cela n'a rien de véritablement étonnant, un système d'exploitation pouvant être considéré à juste titre comme un SGBD gérant les fichiers et les fonctions d'un ordinateur.

Les premiers SGBD ne disposaient que d'un langage de commandes, tel le **SQL** *(system querry langage)* d'IBM. Mais ceux vendus actuellement présentent tous la forme graphique, beaucoup plus sympathique, tout en conservant la possibilité d'accepter des commandes écrites : cette faculté est mise à profit en particulier pour réaliser de petits programmes ressemblant étrangement aux fichiers de commandes des systèmes d'exploitation (fichiers *batch* du DOS par exemple).

Dans les premiers SGBD, la *fiche* de base – ou *enregistrement* – était représentée à l'écran sous la forme d'une ou plusieurs lignes de texte. Avec les systèmes graphiques, la fiche est plutôt représentée comme une zone d'écran (une *fenêtre*) divisée en autant de sous-zones qu'il y a de *champs*, avec parfois des couleurs différentes pour chacun d'entre eux. Avec les SGBD les plus perfectionnés, certains champs peuvent recevoir des images. Dans ce cas, l'écran est réellement géré en *mode graphique*, sinon, il est plutôt géré en mode *semi-graphique*.

2 - CREATION D'UNE BASE DE DONNEES

Un même SGBD peut gérer de nombreuses bases, auxquelles on doit donner un nom au moment de leur création. Une fois le nom défini, on doit également décrire la fiche, parfois appelée *enregistrement*, et, pour cela, définir notamment :

– le nombre de ses champs,

– les caractéristiques de chaque champ.

Cette opération correspond à la description de la *structure* de la fiche en langage de programmation. On devra indiquer quel type de données – parmi ceux autorisés – recevra le champ : texte (ou chaîne de caractères), nombre, date, variable logique, éventuellement image, et la taille des données (longueur des chaînes, nombre de chiffres significatifs ...). Après quoi, la **base** est parée pour son remplissage.

On la remplit en faisant défiler les fiches sur l'écran les unes après les autres. Le système présente beaucoup de souplesse et il n'est pas besoin de remplir d'un seul coup tous les champs de chaque fiche.

3 - SELECTION, MODIFICATION, TABLE D'INDEX

La modification, voire la *destruction*, d'une ou de plusieurs fiches ne pose aucun problème. Signalons cependant que la destruction s'opère souvent en deux temps afin de laisser la faculté en cas d'erreur de récupérer la fiche supprimée ; on retrouve là une philosophie classique en informatique : un fichier détruit par l'utilisateur est rangé temporairement dans un fichier de suffixe .BAK, auquel on pourra revenir *(come back)* en cas de repentir. Ce n'est que lors d'une nouvelle modification que la nouvelle version fera perdre définitivement l'antépénultième.

Pour retrouver une fiche, on peut bien entendu les faire toutes défiler dans l'ordre de leur création. Ce procédé est trop laborieux puisque le SGBD a précisément autorisé la création de ces fiches dans n'importe quel ordre. On adjoindra alors à la *base* des **tables d'index**, qui classeront les fiches en fonction du *contenu* d'un champ ou selon d'autres critères, comme on le verra plus tard. Ces tables ressemblent tout à fait à celle située à la fin d'un livre, permettant de retrouver un mot sans avoir à feuilleter tout le volume.

Il est possible de créer **plusieurs** tables d'index sur une même base. Par exemple, un employeur pourra disposer de tables classant ses employés par nom, par âge, par grade et par ancienneté dans l'entreprise.

4 - RECHERCHE MULTICRITERE

Un tel classement par table d'index constitue déjà une forme d'utilisation des données par ordonnancement. Elle est cependant un peu rigide. La recherche *multicritère* constitue un second moyen beaucoup plus perfectionné. Voici en quoi elle consiste.

Il est possible de demander la *sélection* de toutes les fiches dont les champs contiennent des informations répondant à plusieurs critères simultanés. En somme, cela revient à trouver un **sous-ensemble** dans l'ensemble des fiches. Le sous-ensemble cherché sera formé par l'**intersection** et la **réunion** d'autres sous-ensembles répondant chacun à un critère de sélection.

Les lecteurs initiés à la théorie des ensembles n'auront aucune difficulté à saisir ce mécanisme. Pour les autres, nous allons proposer un exemple :

Une compagnie de chemins de fer pourrait avoir constitué une base de données dont chaque fiche décrirait l'un de ses wagons. S'il se présente une demande de transport de poutrelles en béton de 10 mètres de long pour le jour J entre la ville A et la ville B, le gestionnaire du parc demandera au SGBD de lui fournir les fiches de tous les wagons devant remplir par exemple les conditions suivantes :

– être un wagon plat,
– de longueur supérieure OU égale à 10 mètres,
– se trouver aux jours J-1 OU J-2 OU J-3 ...
– en gare de A, OU de villes proches de A,
– ne pas être déjà retenu pour les jours J OU J+1,
– avoir été révisé depuis moins de quatre mois, c.-à-d. le mois *m*-1 OU le mois *m*-2 OU le mois *m*-3 ...

On a écrit en majuscules les conjonctions OU et ET qui représentent des réunions et des intersections entre sous-ensembles. Bien sûr, il y aura *intersection* entre les 6 sous-ensembles définis par chacune des lignes de critères énoncées ci-dessus, des *réunions* s'étant opérées au préalable dans la plupart des lignes. Les opérateurs disponibles seront en général OR pour la réunion, AND pour l'intersection et NOT pour le complément (ou le contraire).

Dans les SGBD non graphiques, l'opérateur commence par demander la sélection de sous-ensembles reposant sur **un seul critère** de sélection. Le SGBD attribue alors un numéro à ce groupe et en donne le *cardinal*, i.e. le nombre d'éléments trouvés. L'utilisateur définit ensuite étape par étape les opérations logiques à réaliser entre ces sous-ensembles initiaux pour aboutir à celui recherché. Dans ces conditions, il n'est pas facile d'utiliser le SGBD sans un minimum de connaissances en logique. Par contre, avec les logiciels graphiques, la convivialité est telle qu'un peu de bon sens suffit à créer les critères multiples définissant directement le sous-ensemble cherché.

C'est cette propriété de **sélection multicritère** de données qui constitue le gros intérêt de telles bases. Le SGBD réalise en quelques secondes un tri qui aurait demandé autrefois des journées de travail à un opérateur humain. Le *travail de fourmi* des enquêteurs, si souvent mentionné dans les romans policiers, ne demanderait que quelques instants à un SGBD (c'est d'ailleurs l'un des atouts de la police scientifique), à condition que tous les suspects soient fichés, que les indices laissés sur place coïncident avec les champs de la base, que le coupable se trouve parmi les suspects ...

5 - EXEMPLE : ARBRE GENEALOGIQUE

La recherche de ses ancêtres étant très à la mode actuellement, on va montrer comment un SGBD pourrait apporter une aide considérable dans cette quête.

Le postulant bien sûr devra fouiner dans les archives pour en extraire le maximum d'informations sur toute personne susceptible de figurer dans sa généalogie. Puis il rédigera une fiche par personne pour constituer sa base de données, avec des champs qui donneront la date et le lieu de naissance, ceux du décès, la situation matrimoniale... et surtout les noms et prénoms. Il pourra même affecter des coefficients (des nombres) à certaines informations pour situer leur taux de fiabilité.

Ce travail laborieux une fois effectué, il sera facile au SGBD d'établir d'abord une *table d'index* par date de naissance, une autre par nom de famille, avec les *réunions* nécessaires entre noms voisins pour tenir compte des variations possibles d'orthographe au fil des générations. Pour ce faire, on peut parfois utiliser les symboles * et **?** , à la place de – respectivement – une chaîne ou un seul caractère dans un nom, pour laisser au logiciel le soin de les remplacer par n'importe quel caractère réel (encore une analogie avec les systèmes d'exploitation).

On pourra demander au SGBD de trouver toutes les fiches d'hommes pouvant être père d'une personne donnée : noms voisins, dates de naissance différant au plus de 70 (?) ans et au moins de – admettons – 17 ans. On rechercherait de même les mères avec une vérification sur leur situation matrimoniale pour ce qui concerne le changement de nom et en modifiant quelque peu les écarts entre dates de naissance (par exemple 13-55 ans). Gageons que le nombre de fiches sélectionnées par le SGBD sera alors très faible, mais celle du parent recherché y figurera sûrement, du moins si ce parent est bien fiché dans la base et si les critères ont été correctement choisis.

6 - BANQUE DE DONNEES

Le terme de *banque de données* est à l'heure actuelle suffisamment employé pour que nous tentions de le démarquer de celui de *base de données*. Ces mots désignent tous deux des fichiers exploitables par un SGBD. Celui de **base de données** se rapporte plutôt à un fichier dont créateur et exploitant sont identiques. Il en est le propriétaire. Ce propriétaire pourra être une personne morale, c'est-à-dire une société, et donc intéresser plusieurs individus, mais c'est l'identité entre créateurs et exploitants qui définit le mieux une base de données normale.

Au contraire, la **banque de données** est créée par une personne (physique ou morale) pour être mise à la disposition d'un grand nombre d'autres utilisateurs, voire du public. Elle possède une envergure en général bien supérieure à une base de données, mais ce qui la caractérise le plus est la rigueur nécessaire à son exploitation.

Rigueur pour le propriétaire-créateur qui doit respecter le contrat le liant aux utilisateurs de sa banque, d'une part en ne procédant pas à des modifications irréfléchies qui en compromettraient l'exploitation, d'autre part en complétant les données fichées au fur et à mesure de leur évolution.

Rigueur également à l'exploitation : en effet, l'utilisateur ne doit pouvoir ni modifier les données fichées, ni gêner un autre utilisateur de la même banque, ni dépasser les *droits* que lui confère son *contrat*.

Les banques de données sont presque toutes connectées à un *réseau* par l'intermédiaire d'un ordinateur de gestion appelé *serveur*. Par exemple, beaucoup de banques de ce type sont accessibles au public par le réseau *Minitel*. Des banques de données spécialisées sont disponibles pour les *médecins* (informations sur les publications des chercheurs, sur les nouveaux médicaments, statistiques médicales...). D'autres banques offrent des services voisins aux *chimistes* et aux *physiciens* : recherche de publications par thèmes, par auteur, par date ou par laboratoire. De même les *inventeurs* peuvent effectuer leur recherche d'antériorité grâce aux banques fichant les brevets.

Ces banques de données, souvent internationales, sont presque toujours exploitées avec un langage de commandes très peu convivial et même parfois abscons. Leur utilisation est toujours facturée et nécessite abonnement, identificateur et mot de passe, mais leur emploi est rentable pour les spécialistes concernés, à condition de bien savoir s'en servir, malgré le coût élevé de la minute de consultation.

Les sociétés gérant ces banques offrent souvent la possibilité d'obtenir une copie de leur données sur disque optique (CD-ROM). Cette copie reflète l'état de la banque à un instant donné. Elle doit en général être remise à jour périodiquement pour rester d'actualité, ce qu'on obtient en pratique par un abonnement. L'acheteur peut ensuite consulter à sa guise le disque et ses informations. Par ce moyen, les données sont certes moins récentes que celles obtenues par réseau, mais leur consultation est beaucoup moins coûteuse si elle est fréquente, bien que le prix de ces disques soit très élevé et les rende peu abordables aux particuliers.

7 - REMARQUE

L'aisance avec laquelle un logiciel de SGBD manipule une base de données pourrait faire croire qu'il est facile de créer de tels fichiers. Il est en réalité beaucoup plus facile de créer des fichiers quasi inutilisables, voire des fichiers qui se détruisent petit à petit au cours des modifications successives.

En effet, quelques notions théoriques sont indispensables pour un tel travail, qui jusqu'ici faisait appel à du personnel qualifié (les *documentalistes* par exemple). Il existe des livres traitant ce sujet, mais ils sont pour la plupart difficiles à lire.

Le candidat à la création d'une *base* sait en général fort bien ce que seront ses **enregistrements**, ses *lignes* si l'on veut. Ce sont les divers champs de ces lignes qui demandent une réflexion approfondie.

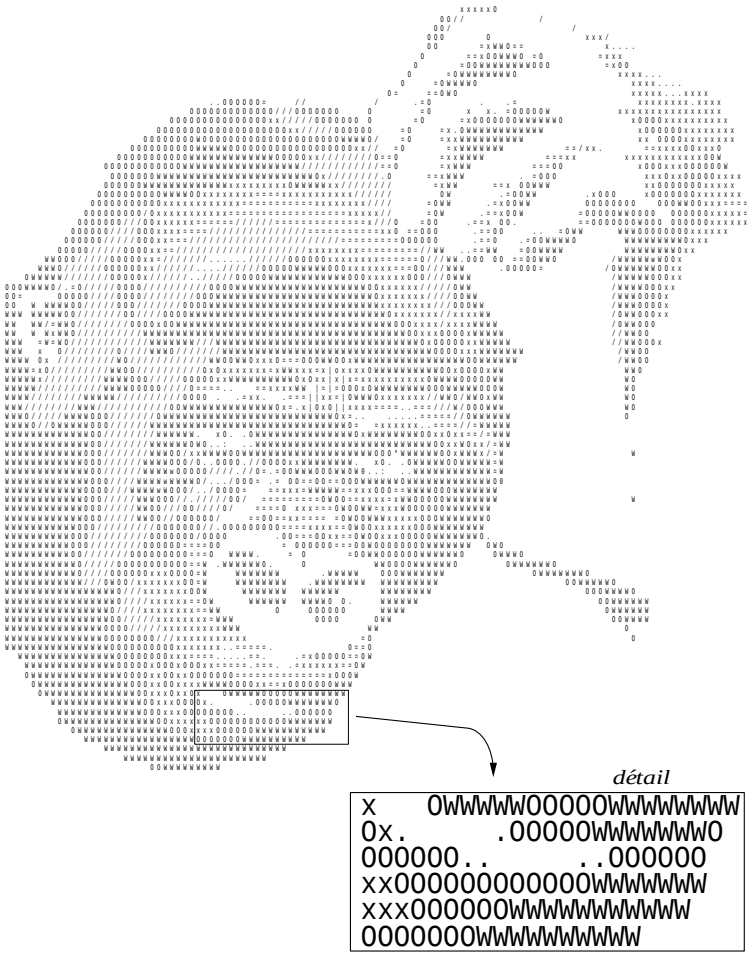
Tout d'abord, il doit exister un champ, appelé **clé primaire**, associé à chaque enregistrement par une relation *biunivoque* (il doit donc être simple et caractérisera l'enregistrement sans aucune ambiguïté). Donnons quelques exemples. Un fichier regroupant les employés d'une société ne peut guère avoir pour clé primaire le nom de ces employés, même associé au prénom ; il y a beaucoup de Martin et de Dupont par exemple, et même de Jacques Martin et de Pierre Dupont. De même il se peut qu'une femme change de nom. Pour toutes ces raisons, on utilise plutôt comme clé celle qu'a inventée l'INSEE pour caractériser tout citoyen, le numéro national, celui utilisé par exemple par l'Armée et la Sécurité Sociale.

De même, le fichier des ouvrages d'une bibliothèque serait peu praticable si sa clé primaire était formée de leur titre (l'article éventuel en tête étant peu significatif et pas toujours cité) ou même de leur cote en rayon, le classement pouvant varier au gré des bibliothécaires. Depuis longtemps, on choisit comme clé primaire pour de tels fichiers le numéro d'entrée des livres dans l'établissement.

Il ne faudrait pas déduire de ces exemples que la clé primaire doit toujours être formée d'un nombre. On peut trouver des contre-exemples. Ainsi, l'une des plus grandes bases qui soient, l'annuaire téléphonique, n'a rien à faire du numéro INSEE qui le surchargerait inutilement. On ne prend pas non plus le numéro de téléphone comme clé puisqu'il caractérise mal un abonné, étant modifié trop souvent. C'est l'ensemble nom-adresse de l'abonné qui a été choisi comme clé primaire. Elle est cependant loin d'être parfaite, vu qu'il est difficile de retrouver dans l'annuaire un abonné ayant déménagé.

On démontre que le choix des autres champs (champs secondaires) n'est pas non plus trivial. En particulier, on doit éviter de prendre des entités qui devraient rester vides (ne recevoir aucune valeur) dans un grand nombre d'enregistrements ou bien qui devraient entrer dans plusieurs champs simultanément : de tels champs alourdiraient ou même mettraient en échec les procédés de sélection multicritère. On donne souvent la règle d'or suivante : *tout champ doit dépendre de la clé, de toute la clé, rien que de la clé.*

On ne s'étendra pas plus sur cette question qui n'est pas vraiment du ressort de l'informatique. Le postulant à la création de bases de données fera bien de se limiter au début à des bases simples et de faible envergure, à moins de s'initier à leur théorie. Mais peut-être verra-t-on bientôt des SGBD capables d'analyser une base mal constituée et de conseiller son auteur sur sa réorganisation.



Gratte-toi la tête, c'est là qu'est le génie ...
Dessin informatique en mode caractère (années 65-70)

Chapitre XIII

L'INFOGRAPHIE

Infographie est un néologisme signifiant **dessin informatique**. La sagesse populaire prétend qu'*un bon dessin vaut mieux qu'un long discours*. Il faut reconnaître qu'on se souvient plus facilement d'une image que d'un texte ou, pis encore, d'une liste de nombres. Dès les débuts de l'informatique, on a cherché à tracer des dessins. Les premiers étaient composés avec les caractères de l'écriture courante. On a ainsi créé de véritables tableaux monocolores, simulant le relief par la différence de densité des caractères : pour obtenir des zones d'ombrage croissant, on les remplissait avec des points, puis avec des lettres I, puis des L, ... pour terminer par des W. Ces dessins devaient être regardés d'assez loin : on dit qu'ils possédaient une *très mauvaise résolution*.

Aujourd'hui, les imprimantes à aiguilles permettent de placer des points n'importe où sur le papier avec une *résolution* de l'ordre de 3 à 5 par millimètre. Les imprimantes à laser de bureau atteignent couramment 300 points par pouce (environ 12 par millimètre). Mais on dessine également beaucoup à l'**écran**, qui présente l'avantage de la **couleur**, avec toutefois l'inconvénient d'une résolution moindre.

On pourrait diviser ce chapitre en fonction des périphériques employés (écran, imprimante, traceur ...), du mode de tracé en service (image de points, vectoriel) ou du langage utilisé (haut niveau, HPGL, PostScript). Nous préférerons adopter un schéma progressif, partant des procédés les plus simples pour aboutir aux plus perfectionnés.

1 - DESSIN EN MODE CARACTERE

Ce mode de dessin est identique à celui des **premiers temps** de l'informatique. Mais on disposera de moyens inconnus alors : écrans couleur, imprimantes à haute résolution et caractères spécialement conçus pour le graphisme. Pour dessiner à l'écran avec les seuls caractères d'imprimerie, il faut savoir comment amener le curseur à un emplacement quelconque, ce qui n'est pas si simple, puisque le DOS ne sait pas le faire. Cela s'appelle gérer l'écran en *mode écran* par opposition au *mode ligne*. Il faut savoir aussi maîtriser la *couleur* et utiliser les quelques caractères adaptés au dessin.

a - Séquences d'échappement

Déplacement du curseur, soulignement, surbrillance, couleur de l'écran se gèrent sous DOS par des suites de caractères appelées *séquences d'échappement ANSI*. Elles **commencent** par le caractère ASCII n°27, *Esc* (*escape*), facilement introduit en langage évolué (**CHR$(27)** par ex. en Basic), mais très difficilement sous DOS, la touche *Esc* étant *interprétée* comme caractère de commande (on peut essayer les combinaisons de touches *Alt* **27** ou *Ctrl* **[**, ou **$e**, cette dernière ne fonctionnant guère qu'après la commande **prompt**. En désespoir de cause, on introduira ce caractère sous **DEBUG**, cf. page 67). Ces séquences ou **consignes** ne sont opérationnelles que si l'on a chargé en mémoire leur *interpréteur*. On inclura dans le fichier CONFIG.SYS la ligne de commande

device = *(répertoire)* **\ ANSI.SYS**

Voici quelques séquences d'échappement et leur rôle:

Esc **[2J**	efface l'écran
Esc **[***x***;***y***H**	amène le curseur en *x, y*
Esc **[33;44m**	caractères jaunes sur fond bleu
Esc **[7m**	caractères blancs sur noir *(vidéo inverse)*

Les séquences d'échappement régissant la couleur (caractères et fond) restent actives jusqu'à contrordre. Tous les logiciels n'acceptent pas ces séquences : elles requièrent de la prudence dans leur manipulation et ne concernent que l'écran. Leur envoi sur une imprimante entraînera en général des effets perturbateurs, comme le changement de page ou celui de police, car la mise en forme de l'impression s'obtient également par des séquences d'échappement **(1)**.

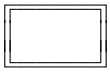
b - Caractères graphiques

Toujours sous DOS, le jeu de caractères appelé *extension* IBM à l'ASCII ou **jeu semi-graphique IBM** permet de tracer lignes droites, rectangles, cadres,

(1) Autrefois, cette mise en page s'obtenait par l'*interprétation* du premier caractère de chaque ligne, qui n'était pas imprimé, mais jouait le rôle d'un code de commande.

organigrammes. Ces caractères sont représentés avec leur numéro (**supérieur à 128**) dans le tableau ASCII de l'annexe 1. Ils assurent le rendu correct de **lignes horizontales ou verticales**, simples ou doubles, avec des intersections jointives, comme dans la fig. 6-2 page 44. Pour les mettre en œuvre, en principe il faut charger un *programme résident* (c.-à-d. demeurant en permanence en mémoire, tel ANSI.SYS) capable de les interpréter. On inclura la ligne de commande **GRAFTABL** dans le fichier AUTOEXEC.BAT. Non disponibles au clavier, ces caractères sont introduits par leur numéro qu'on frappe avec la touche ALT enfoncée. Ainsi, le cadre ci-après est dessiné par les trois lignes de *consignes* suivantes:

Alt 201, Alt 205, .. 205, .. 205, .. 205, .. 187
Alt 186, quatre espaces, Alt 186
Alt 200, .. 205, .. 205, .. 205, .. 205, .. 188



Si **GRAFTABL** est actif, la frappe de ces consignes produira à l'écran les caractères graphiques désignés. Ils peuvent être placés dans un fichier grâce à un éditeur ASCII. Lorsque ce fichier sera affiché à l'écran (par ex. avec la commande **TYPE** du DOS), ces caractères seront correctement interprétés. Ils peuvent également être incorporés à des *chaînes* placées en argument dans des ordres d'**affichage** tels que **PRINT, WRITE, DISP, cputs** ... en langage évolué (il en est de même pour les séquences d'échappement).

Comme on l'a vu dans le chapitre précédent, certains **tableurs** autorisent l'emploi des caractères semi-graphiques pour dessiner le *quadrillage* s'il n'est pas généré automatiquement. Par contre, il est rare de pouvoir les utiliser aussi directement avec les traitements de texte élaborés, surtout ceux utilisant des polices proportionnelles. Certains d'entre eux possèdent une police spéciale comprenant ce genre de caractères. Leur emploi exigera peut-être de passer en *police fixe*, pour garantir l'alignement des segments de ligne.

Ce procédé graphique est très employé par les concepteurs de logiciels pour en agrémenter la présentation et écrire les *menus*. Mais on est limité à l'angle droit ! Ces dessins présentent l'avantage d'être **reproductibles sur papier** par différents moyens : par ex. par la frappe de la touche *Impr-écran* (à condition d'avoir auparavant chargé un résident lancé par cette touche – **GRAPHICS** sous DOS, voir aussi note 5, p. 118). Un autre procédé consiste à programmer l'impression de chaînes contenant ces caractères semi-graphiques ainsi que les séquences d'échappement nécessaires (on rappelle qu'elles sont différentes de celles prévues pour l'écran).

2 - DESSIN POINT A POINT SUR ECRAN

a - Initialisation.

Comme on l'a vu dans le chapitre V, le *mode graphique* permet de gérer tous les pixels de l'écran et non plus seulement les quelque 80×25 caractères habituellement disponibles en *mode caractère*.

Pour passer en mode graphique (cf. chap. V, §3bγ, p. 38), il faut commuter des circuits dans les cartes gérant l'écran, ce qui s'obtient par une *interruption* (celle de numéro 10h-0 sous DOS) ou par l'instruction prévue à cet effet dans les langages évolués (**INITGRAPH, SCREEN**, ...). La **carte graphique** obéit à des **normes** qui la rattachent à un certain nombre de *modes graphiques* définis par IBM, comme **CGA**, **EGA**, **VGA**, ... ou par d'autres comme **Hercules**, **SuperVGA**. Il est extrêmement conseillé, avant de commencer à programmer un dessin, de savoir exactement de quel mode on dispose. Les langages les plus perfectionnés possèdent des instructions permettant de demander à la machine de quelle carte graphique elle est munie et quelle est sa résolution [2]. Il faut savoir que ces modes sont repérés par un numéro et que la plupart des cartes acceptent de travailler sur demande expresse selon plusieurs modes de numéro au plus égal à celui qui les caractérise.

On ne détaillera pas la création de dessins en **langage machine**, ce procédé étant peu utilisé. Il consiste à placer les bits convenables (instruction **MOV**) dans des octets, images des pixels de l'écran. Ces octets constituent un sous-ensemble de la mémoire vive, appelé **mémoire vidéo**, située au-delà des 640 K accessibles au chargeur du DOS. Plusieurs images d'écran, appelées **pages**, peuvent être disponibles et programmables, mais une seule – la *page visuelle* – est à un instant donné convertie en image écran.

Presque tous les langages évolués permettent une gestion fine de ce mode graphique, mais nous ne donnerons d'exemples qu'en **Basic**. Dans ce langage, on ouvre le programme de tracé par l'instruction

SCREEN *mode*, , *n*, *m*

où *mode* (de 1 à 13, 0 étant le *mode texte*) sélectionne le mode graphique (il faut connaître sa valeur ou procéder par essais) [3], *n* et *m* (facultatifs) les numéros éventuels de la page *active* (page dans laquelle on écrit) et de la page *visuelle*.

(2) De tels ordres, à condition de tenir compte des informations reçues, sont indispensables pour rendre un programme graphique *portable*, c'est-à-dire indépendant de la machine qui l'exécute.

(3) **SCREEN** est une instruction délicate, dont les possibilités dépendent beaucoup du matériel (carte vidéo) et du Basic utilisés.

b - Fenêtres et couleurs

On peut limiter le graphique à une fenêtre rectangulaire dont la diagonale est définie par les points $\{x_0, y_0\}$ et $\{x_1, y_1\}$, grâce à l'instruction

VIEW $(x_0, y_0) - (x_1, y_1), c_1, c_2$

Les entiers c_1 et c_2 attribuent les couleurs c_1 et c_2 respectivement au tracé et au fond de l'écran. Les coordonnées x_0, y_0, x_1, y_1 sont ici en *notation absolue* et correspondent au rang des pixels (par exemple en mode VGA, x varie de 0 à 639, y de 0 à 479, le point {0,0} étant situé **en haut à droite** de l'écran).

On réoriente cette fenêtre et on définit de **nouvelles unités** en fixant des valeurs pour ses sommets, le tout grâce à l'ordre

WINDOW $(X_0, Y_0) - (X_1, Y_1)$

où les coordonnées $\{X_0, Y_0\}$ désignent maintenant le point en bas à gauche de l'écran (convention plus traditionnelle). Par exemple, les valeurs X_0=Y_0=0 et X_1=Y_1=100 permettront d'exprimer ensuite toutes les positions comme des pourcentages des dimensions de la fenêtre avec l'orientation classique et les rend, dans **la suite** du programme, indépendantes de la carte graphique.

La **couleur** est fixée par des arguments propres à chaque ordre graphique, ou à défaut par un ordre général fixant la *couleur courante* c_1 :

COLOR c_1

Le comportement de **COLOR** dépend beaucoup du matériel. La couleur du tracé à venir sera c_1. Voici la correspondance par défaut entre couleur et numéro c_i :

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
noir	*bleu*	*vert*	*cyan*	*rouge*	*violet*	*orange*	*gris*
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
gris	*bleu*	*vert*	*cyan*	*rouge*	*violet*	*jaune*	*blanc*

(les couleurs 0...6 et 8 sont **foncées**, les autres **claires**; le *cyan* est un bleu-vert). L'ordre **PALETTE** modifie instantanément la couleur de l'un de ces numéros, même pour les dessins déjà affichés, fond y compris.

Avec les meilleures cartes graphiques, on peut créer un tableau retenant 16 ou même 256 couleurs parmi 262 000 offertes. L'instruction **PALETTE USING** *tableau*, remplace par celles de *tableau* les couleurs attribuées par défaut aux codes c_i . Chacune de ces nouvelles couleurs est définie à partir des trois composantes fondamentales rouge-vert-bleu en donnant leur *taux r* selon la relation ci-après, pour la *k*ième couleur (*tableau* est formé d'entiers longs) :

$$tableau(k) = 65536 * r_{bleu} + 256 * r_{vert} + r_{rouge}$$

c - Tracé de points et lignes en Basic

Les instructions suivantes permettent de tracer :

– un point normal en x, y : **PSET** (x, y) **[**, c**]**
– un point avec la couleur du fond **PRESET** (x, y)
– une droite, de $\{x_0, y_0\}$ à $\{x_1, y_1\}$
LINE $(x_0, y_0) - (x_1, y_1)$ **[**,c,,k**]**

Le paramètre couleur *c* est placé entre crochets **[ ]** pour indiquer qu'il est **facultatif.** De même pour *k* qui fixe le type de pointillé (on en parlera § **e**). L'instruction **LINE** comporte beaucoup de variantes qui sont résumées par l'écriture rigoureuse ci-après, très employée dans les manuels de programmation :

LINE [[STEP] (x_0, y_0)**] – [STEP]** (x_1, y_1) **[**,[c]**] [**,B **[**F**]]] [**, k**]**

Tous les paramètres entre crochets sont facultatifs. Ceux que l'on omet doivent être remplacés par la virgule qui les précède sauf si on omet également les paramètres qui les suivent. Ainsi, sont acceptées les instructions suivantes :

LINE $(x_0, y_0) - (x_1, y_1)$
LINE – STEP (x_1, y_1) , c ,, k

Dans ces instructions, les coordonnées des points sont celles des deux extrémités de la droite à tracer, exprimées dans le système déterminé par l'instruction **WINDOW** ou, si elle est absente, en notation absolue. Le mot **STEP** transforme les coordonnées adjacentes en coordonnées relatives, c'est à dire qu'elles ont pour origine non pas le zéro de la fenêtre graphique, mais le dernier point tracé (qu'on appelle *point courant* ; x_1 et y_1 deviennent des longueurs). L'omission du premier point (x_0 , y_0) entraîne son remplacement par le point courant.

d - Rectangles, cercles, ellipses, secteurs

Le **rectangle** se dessine simplement grâce à l'instruction **LINE** ci-dessus en utilisant le paramètre **B** pour n'obtenir que sa bordure et **BF** pour obtenir un rectangle plein. Comme toujours, il est déterminé par les extrémités de sa diagonale $\{x_0 , y_0\}$ et $\{x_1 , y_1\}$.

Les figures circulaires et elliptiques s'obtiennent par l'instruction

CIRCLE [STEP] (x_0 , y_0) , r , c , α_1 , α_2 , asp

dans laquelle tous les paramètres sont facultatifs sauf le centre x_0 , y_0 et le rayon r du cercle générateur. On obtient dans ce cas un **cercle** de couleur *c*, ou, si *c* est omis, de la couleur courante.

Si le paramètre *asp* est exprimé (et ≠1), on obtient une **ellipse** de rapport d'aspect *asp*. Le petit axe a pour valeur r ; le grand axe est vertical et vaut *r/asp* si *asp*<1 ; il est horizontal et vaut *r*asp* si *asp*>1 .

Si les angles α_1 et α_2 sont explicités (en radians, avec les conventions trigonométriques) (4), on obtient un **arc de cercle** ou d'ellipse limité par ces deux angles. Si l'un d'eux ou les deux sont négatifs, l'extrémité correspondante est reliée par une droite au centre du cercle. On obtient ainsi un **secteur**, i.e. un morceau du fameux *camembert* (chap. XII).

e - Décoration: pointillé, remplissage, textes

On sait déjà qu'on obtient un rectangle plein avec le paramètre **BF** de **LINE**. De manière générale, on **remplit** de la couleur c_1 **une figure fermée** et, éventuellement, on colorie sa **bordure** avec la couleur c_2, grâce à l'instruction :

PAINT (*x*, *y*) , [c_1 , c_2]

Le point *(x,y)* doit être situé à l'intérieur de cette figure. Si, au lieu d'être un nombre, c_1 est un chaîne, la figure sera remplie avec le *motif* décrit par cette chaîne (se reporter aux ouvrages sur le Basic pour plus de détail). Mentionnons seulement que les chaînes **"U"** ou **CHR$(85)** et **"$"** ou **CHR$(36)** provoquent le remplissage d'une figure par des pointillés.

Les **lignes** et bords de figures sont normalement tracées **en trait plein**. On obtient un tracé **discontinu** en explicitant l'option *k* dans l'ordre **LINE**. L'**entier** *k* , suite de 16 bits, sera considéré comme un motif décrivant le tracé en faisant correspondre tout bit égal à 1 avec un pixel éclairé et tout bit égal à 0 avec un pixel éteint. Ainsi, on obtient un pointillé très fin avec *k*=**&HAAAA**=**43690**, moins fin avec *k*=**&HCCCC**=**52428** ou **&H8888**=**34952**, les tirets avec *k*=**&HFF00**=**65280** et un trait alterné (point-tiret) avec *k*=**&HF6F6**=**63222**.

Une autre instruction – **DRAW** – permet d'exécuter les mêmes tracés avec une écriture plus concise et moins "parlante". Nous ne l'étudierons pas ici.

Du **texte** peut être ajouté au dessin. En Basic, on emploiera les instructions **PRINT** habituelles après avoir placé le curseur à la position convenable *x,y* (coordonnées cette fois-ci en *mode caractère*), grâce à l'instruction

LOCATE *x*, *y*

Les langages perfectionnés autorisent l'emploi de polices spéciales (proportionnelles en particulier), de taille variable et n'importe où sur l'écran (donc en mode graphique), mais le Basic ne permet d'écrire qu'en *mode caractère*. (5)

f - Animation du dessin

Pour simuler le mouvement, on emploie la technique des dessins animés, c.-à-d. la présentation à l'écran d'images successives dont seule diffère la position des objets mobiles. En informatique, l'une des difficultés consiste à régénérer le dessin du fond à la place libérée par les mobiles après leur déplacement.

On pourra utiliser l'aide parfois offerte par des *pages graphiques* multiples. On remplit par les instructions ci-dessus la page active, pendant qu'une autre est affichée à l'écran (pages définies par l'ordre **SCREEN** *k*, , *n*, *m*). En temps voulu, on permute les deux pages par l'ordre ci-après *(n, m* doivent être alors des variables et non des constantes)

SWAP *n*, *m*

La reconstruction d'une page entière étant assez longue, on peut ne travailler que sur la partie évolutive du dessin. Celle-ci est, supposons, délimitée par le rectangle de diagonale $\{x_0,y_0\}$-$\{x_1,y_1\}$. On commence par prévoir pour elle un tableau *(tab)* de taille suffisante (6). On dessine dans ce tableau la nouvelle portion d'image, puis on *plaque* le tableau *tab* sur l'écran, à l'origine $\{x_0,y_0\}$, grâce à l'instruction

PUT (x_0 , y_0), *tab*, *option*

L'*option* est très importante ; elle peut prendre 5 valeurs : **PSET**, **PRESET**, **AND**, **OR**, **XOR** : **PSET** surcharge le dessin par le tableau *tab* ; **PRESET** le surcharge par son inverse (*tab* mis en *vidéo inverse*). Les trois autres options réalisent entre bits correspondants les opérations logiques dont elles empruntent le nom. **AND** réalise un filtrage, **OR** *fond* (mélange) les deux images. **XOR** permet de superposer, puis d'enlever, une image à une autre. C'est l'option à employer pour récupérer l'image du fond libérée par un mobile. Ainsi, le tableau *mob* représentant une fraction d'image peut être *ajouté* puis *retiré* de l'écran par deux instructions successives et identiques comme

PUT (*x*, *y*) *mob*, **XOR**

L'instruction suivante, symétrique du **PUT**, copie en mémoire dans le tableau *mob* une zone rectangulaire de l'écran :

GET [STEP] (x_0, y_0) – **[STEP]** (x_1, y_1), *mob*

Tels sont les moyens qu'offre le Basic pour produire du dessin animé. Les autres langages ont, dans ce domaine, des possibilités à peine supérieures.

---

(4) Les conventions trigonométriques sont les suivantes : sens inverse de la rotation des horloges et origine des angles à la position "3 heures".

(5) Sous DOS4 et ultérieurs, on peut imprimer un écran en mode graphique avec les touches MAJ-Impr-écran si on a chargé auparavant **GRAPHICS** avec, en paramètre, le code de l'imprimante (ex. **/larserjetii** pour une laserjet II).

(6) Le nombre d'octets nécessaires est $h*p*\mathrm{Int}[(l*b+7)/8]$, où *h* et *l* sont les dimensions en pixels du rectangle, *b* le nombre de bits par pixel (1 en monochrome, ... 4 en VGA) et *p* le nombre de plans couleurs (1, 2 ou 4 selon le mode). Les langages perfectionnés ont des instructions capables de calculer automatiquement cette taille.

3 - DESSIN VECTORIEL SUR TABLE TRACANTE

La table traçante, ou *traceur (*anglais *plotter*), est un périphérique qui déplace une *plume* sur une **feuille de papier**. Couleur de l'encre et position de la plume peuvent être programmées en fonction du matériel disponible. La plume étant entraînée par des *moteurs pas à pas*, le tracé est très fin : sa précision est de l'ordre de 0,1 mm et sa résolution (valeur du plus petit pas programmable) est de 50 ou même 25 microns. Certaines imprimantes peuvent également se comporter comme un traceur : elles sont bien plus rapides, mais souvent monocolores et limitées au papier A4.

La table n'accepte qu'un langage spécial, très souvent le **HPGL** *(Hewlett-Packard graphic language)*. Un *interpréteur* de commandes est logé dans la table au sein d'un *processeur* couplé à une *mémoire-tampon (buffer)*. La table attend les *commandes* – ou *ordres* – HPGL sous forme de chaînes de caractères ASCII; les mouvements de la table étant beaucoup plus lents que l'envoi des ordres, la mémoire tampon les *accumule*. Mais, de faible capacité, elle est vite saturée ; elle envoie alors un signal au calculateur via le câble qui les relie. Si le programme ne tient pas compte de cet avertissement et continue à expédier des ordres de tracé, le dessin sera erroné.

Les commandes HPGL commencent par un mot-clé *(mnémonique)* formé de deux lettres, suivi d'arguments parfois facultatifs, mais qui doivent être des **constantes** (7) séparées par un espace ou une virgule. Une *instruction* peut comporter plusieurs commandes à la suite, même sans signe de séparation entre elles. Elle *doit* se terminer par un **point-virgule**. Chaque traceur ne dispose pas toujours de la totalité des commandes. Seules les plus importantes seront citées ici.

Vues du côté calculateur, les commandes HPGL constitueront des *chaînes* qui seront envoyées au traceur via un ordre de sortie (**OUT** en Basic) à l'adresse de la carte gérant ce périphérique. Plus simplement, mais moins puissamment, on peut écrire ces chaînes dans un *fichier* de numéro *nf* :

PRINT #*nf, chaîne*

où *chaîne* est l'instruction HPGL. Ce procédé (mode *fichier*) permet d'envoyer au traceur le contenu de variables de manière très simple. On écrira par exemple :

PRINT #9, **"IN, DT£, IW",** x_1 , x_2 , y_1 , y_2 , **";"**

où les mots entre guillemets font partie du langage HPGL. Les noms x_1 ... y_2 désignent des variables dont la fonction **PRINT** extraira la valeur pour l'envoyer dans le fichier n° 9. Elle sera interprétée comme un argument du mnémonique **IW**.

(7) Au sens informatique du terme, c'est-à-dire pas des noms de variables, mais des valeurs numériques, même réelles, ou des chaînes.

a - Initialisation

Il faut d'abord initialiser la liaison entre traceur et calculateur, liaison qui peut revêtir trois formes :

– **liaison parallèle Centronix**, rarement disponible sur les traceurs, mais très simple à utiliser. Elle monopolise l'unique sortie d'imprimante (sauf si on possède une carte imprimante LPT2 ou 3). En Basic, il suffit alors de travailler en *mode fichier* (de numéro *nf)* après l'avoir ouvert grâce à

OPEN "LPT*m***:" FOR OUTPUT AS #***nf* (*m*= 1 2, ou 3)

– **liaison série** ou RS232, dont la gestion précise est trop complexe pour être détaillée ici (elle fera l'objet d'une étude dans l'annexe 2). Si l'on se borne au *mode fichier*, on ouvrira la liaison en Basic par

OPEN "COM1: *d, p, n, a, options***" AS #***nf*

où le numéro 1 de **COM** sera 2, 3 ou 4 selon la sortie série utilisée. Les variables numériques *d* (débit, en bauds), *p* (parité : **E** pour pair – *even* –, **O** pour impair – *odd* –, **N** pour néant – *null* –), *n* (nombre de bits par caractères : 7 ou 8) et *a* (nombre de "bits" d'arrêt: 1, 1.5 ou 2) doivent être de même valeur que celles du traceur (sur ce dernier, ces variables sont parfois ajustables par des commutateurs. Attention, certains traceurs ne permettent pas d'ajuster le nombre de bits par caractère, qui est alors égal à 7).

– **liaison GPIB** (ou IEEE-388), auquel cas il faudra suivre la procédure indiquée dans le mode d'emploi du logiciel qui gère cette ligne (en général, on passe également par un fichier). Il faudra attribuer une adresse GPIB au traceur grâce à un commutateur, laquelle adresse devra, d'une façon ou d'une autre, être communiquée au calculateur.

Les liaisons *Centronix* et GPIB garantissent une transmission sans débordement de la mémoire tampon grâce à une **concertation** *(handshake)* entre ordinateur et traceur. Mais avec la *liaison série*, il faudra (à moins d'une faible quantité de dessins) se donner le moyen d'un tel contrôle. Les ordres afférents se placeraient à ce stade (le lecteur confronté à ce problème se reportera utilement à l'annexe 2).

Il faut ensuite, dans tous les cas, prévoir quelques initialisations complémentaires, comme :

– **IN ;** qui remet tous les paramètres de la table à leur valeur par défaut.

– **DT** (optionnel) qui permet de fixer le caractère terminateur de chaîne (nous conseillons **DT£** , le symbole **£**, facile à taper, étant inutile en français. Sinon, le symbole pris par défaut est le caractère ASCII n°3, soit Alt 3).

b - Fenêtres et couleurs

Beaucoup de traceurs possèdent plusieurs plumes, chacune rangée en bordure ou dans un *barillet*, à l'emplacement *n*. Le traceur prend la plume *n* sur l'ordre HPGL **"SP",***n*,**";"**. **"SP0;"** fait ranger la plume en service. C'est le seul moyen disponible pour modifier **couleur** ou **épaisseur** du trait en disposant les plumes *ad hoc* au bon emplacement.

Le traceur possède un repère de coordonnées *propre* : l'origine en est un coin du papier et l'unité le pas des moteurs (*repère machine*). Ce repère peut être redéfini. On sélectionne d'abord une fenêtre par

PRINT #*nf*, **"IP",** x_0, y_0, x_m, y_m **,";"**

où les *x,y* sont exprimés dans le *système machine* ($x_0<x_m$ et $y_0<x_m$). On calque ensuite sur cette fenêtre, grâce à **SC**, un nouveau système d'unités. Avec un traceur HP à rouleau et de format A4, on écrira

PRINT #*nf*, **"SC",** Y_m, X_0, Y_0, X_m **,";"**

pour amener l'origine du nouveau système de coordonnées X_0 Y_0 X_m Y_m en bas et à gauche de la fenêtre choisie (disposition traditionnelle). Les unités de ce nouveau système peuvent par exemple être des centimètres, ce qui rend les unités bien plus sympathiques que des coordonnées machine.

Toutes les coordonnées subséquentes devront être exprimées dans le nouveau système. On peut également programmer une fenêtre de *masquage*, qui limitera la zone du tracé à un rectangle dont les diagonales sont définies par les points X_1 , Y_1 , X_2 , Y_2. L'ordre à donner est :

PRINT #*nf*, **"IW",** X_1, Y_1, X_2, Y_2, **";"**

c - Tracé des figures

Dans ce mode, il n'est pas question de tracer des *points*, mais seulement des segments de ligne droite, des *vecteurs* (d'où son nom). Pour tracer une **droite** ou déplacer la plume, 4 mots-clés sont prévus :

"PD" qui abaisse la plume sur le papier,
"PU" qui la relève,
"PA" qui interprète les positions comme des coordonnées absolues,
"PR" qui les interprète comme valeurs relatives.

Ces ordres restent en vigueur jusqu'à l'ordre contraire. Ces quatre mots-clés *peuvent* être suivis de deux valeurs *x* et *y*, coordonnées du point où se déplacera la plume : si le dernier de ces ordres précédant *x* et *y* est **"PA"**, *x* et *y* seront des coordonnées *absolues* (éventuellement dans le système défini par **"SC"**) ; elles seront *relatives* au dernier point tracé *(point courant)* si le dernier ordre les précédant est **"PR"**.

Les ordres ci-après tracent **rectangles, cercles**, **arcs** et **secteurs**. S'ils se terminent par **A**, leurs deux arguments seront considérés comme des coordonnées *absolues* ; s'ils se terminent par **R**, elles seront *relatives* (on accentue ici cette convention en faisant précéder de *d* les noms de coordonnées relatives). Les angles doivent être donnés en degrés avec les conventions trigonométriques classiques.

EA *x, y* — **ER** *dx, dy* — rectangles.
RA *x, y* — **RR** *dx, dy* — rectangles pleins.
CI *r, ang* — polygone fermé centré sur le point courant, de rayon *r* et de pas angulaire *ang*.
AA *x, y, a,* [,*b*] — arc de cercle débutant à la position actuelle, centre *x, y*, ouverture *a*, pas angulaire *b*.
AR *dx, dy, a,* [,*b*] — idem, mais relatif.
EW *r,* a_1, a_2, [,*b*] — secteur de rayon *r*, d'angle initial a_1 (par rapport à la position actuelle), d'ouverture a_2, de pas *b*.
WG . . . . — idem, mais avec remplissage.

e - Pointillé, remplissage, textes

L'instruction "**LT**", *n* [, *p*],**";"** provoque le tracé en **pointillé** ou en *tireté* des segments de droites ultérieurs jusqu'à contrordre (par exemple par "**LT;**"). *n* (égal au plus à 6) désigne le type de motif et *p* sa période (en pourcentage de la diagonale de la fenêtre).

Le **remplissage** n'est possible qu'avec les mots-clés **RA**, **RR** et **WG** ci-dessus. On choisit auparavant le motif par **FT** *n,* **[,** *p***[,** *a***]]** où l'entier *n* donne le type du motif et *p* sa période (comme avec **LT**).

L'écriture de textes s'obtient par l'instruction **PRINT** #*nf* suivie de

"LB texte reproduit£" ou par **"LB ",** *chaîne*, **"£"**

où le texte suivant **LB** sera reproduit *tel quel*. Si ce texte est contenu dans une chaîne, il faut écrire l'instruction comme ci-dessus à droite. Dans chaque cas, il faut signaler à l'interpréteur la fin du texte à écrire par le caractère *terminateur* de texte, supposé ici être **£**. Les instructions suivantes affectent les écritures :

CS *n* modifie le jeu de caractères,
SI *l, h* en cm, fixe largeur et hauteur des caractères,
DI *dx, dy* donne l'angle θ de la ligne, avec tgθ = *dx/dy*,
CP ±*n*,±*m* déplace la plume de *n* caract. et *m* lignes,
SL *t* fixe l'inclinaison θ des caractères (*t* = tg θ).

Il est possible également de modifier la vitesse de la plume (**VS** *v*, *v* en m/s), constituer son propre jeu de caractères (**UC** *tableau*), trouver la position de la plume (pour *numériser* des points). Ces fonctions ne seront pas décrites ici. On rappelle que tous ces ordres HPGL doivent faire partie d'une liste d'arguments dans un ordre **PRINT** #*nf* .

Chapitre XIV

L'INFOGRAPHIE (2e partie)

1 - DESSIN VECTORIEL EN LANGAGE POSTSCRIPT

Pour programmer du dessin de haute qualité, on pourra employer PostScript (PS). Conçu par *Adobe Systems Inc*., ce langage était destiné aux *développeurs* de logiciels d'impression et de mise en page, intermédiaires entre d'une part les imprimantes laser ou les photocomposeuses et, d'autre part, les meilleurs TTX ou les logiciels de PAO. La qualité des impressions des Macintosh doit beaucoup à PostScript.

Les utilisateurs d'imprimantes PostScript peuvent envoyer à ces machines des fichiers correctement rédigés qui ne seront pas imprimés tels quels, mais verront les commandes qu'ils contiennent interprétées pour générer impressions ou dessins. C'est là une analogie avec le langage HPGL, puisque l'imprimante PostScript dispose d'un interpréteur et d'une mémoire (de grande capacité, cette fois).

Ce langage est difficile pour plusieurs raisons : il est **puissant** et la puissance se paye toujours par la **complexité**. Ensuite, contrairement au HPGL, il est **procédural**, i.e. il permet la décomposition du programme en des *procédures* multiples. Enfin, il emploie la *notation polonaise inverse* (comme dans les calculettes HP, cf. p. 58, note 6) et transmet ses arguments par l'une des **piles** qu'il manipule.

Comme le fait implicitement HPGL, PostScript emploie la notion d'**état graphique**, englobant le type de trait (pointillé, épaisseur, forme des extrémités, des intersections), le type de remplissage *(grisé)*, la couleur ... On peut *mettre en réserve* cet état dans une pile spéciale pendant l'exécution d'une procédure nécessitant un autre état graphique.

Autre notion délicate : celle de **chemin** *(path)*, suite de segments initialisée par certains ordres (**newpath**, **moveto**), définis par d'autres (**lineto**, **rlineto**, **arc**, **arcto**, ...). On pourra le fermer (**closepath**), le remplir (**fill**) ou le charger d'un trait (**stroke**) défini dans l'état graphique. Le chemin peut comprendre les fameuses *courbes de Bézier* **(1)** (**curveto**) ainsi que les caractères des nombreuses polices disponibles (**charpath**).

Les coordonnées des points tracés, arguments des ordres de chemin, sont par défaut celles d'un **système d'axes** lié au papier et calqué sur celui des imprimeurs : l'**origine** {0,0} est **en bas à gauche** de la page et l'unité le *point* (1/72e de pouce). Ce repère est constamment modifié par la *matrice de coordonnées*, grâce à laquelle on peut demander dilatation (**scale**), translation (**translate**), rotation (**rotate**) ou des opérations plus complexes comme la symétrie. L'existence de cette matrice permet de définir à part des objets (ou des caractères) puis de placer lesdits objets, agrandis, réorientés, en n'importe quel endroit de la page.

Toutes les **polices de caractères** de PS sont vectorielles, c.-à-d. définies comme des courbes et dilatables à volonté par la matrice de coordonnées. Un très grand nombre de tailles (*corps*) sont donc disponibles. Il est possible de redessiner ses propres polices, mais il s'agit là plutôt d'œuvre de spécialistes.

Ce langage reconnaît toutes les opérations arithmétiques (**add**, **sub**, **mul**, **div**), quelques fonctions mathématiques (**sin**, **log**, **exp**, ...), les opérateurs de comparaison et ceux d'algèbre de Boole. Il manipule les tableaux, les chaînes et les fichiers. Exécuté en séquence avec de très nombreux appels de procédures, il accepte des blocs de type **boucle** (**for**, **forall**, **repeat**, **loop**) ou dépendant de conditions (**if**, **ifelse**).

Chacun des 259 mots-clés de PS exige un certain nombre d'arguments qu'il prélève sur la *pile de travail* (LIFO). Il faudra les avoir placés auparavant dans le bon ordre. On peut manipuler cette pile : retirer l'élément supérieur (**pop**), le dupliquer (**dup**), échanger les deux premiers (**exch**) et d'autres plus profonds (**roll**).

PostScript permet de programmer des schémas de haute qualité. Il gère les parties cachées et s'avère très efficace dans la manipulation des images échantillonnées (*bitmap*) : modification de taille, de contraste... Certaines imprimantes compatibles PostScript se commutent automatiquement dans le mode PS dès qu'elles reçoivent la séquence **%!**. Pour mettre au point ces programmes, on peut utiliser **GhostScript**, logiciel *en libre diffusion*, qui les représente à l'écran avec rapidité et précision. Ce sont des moyens puissants qui justifient l'apprentissage de PS, vu la qualité des services rendus et la faiblesse de leur coût **(2)**.

---

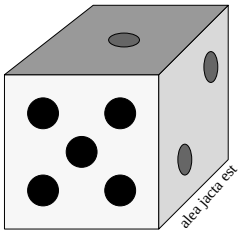
(1) Courbes du 3e degré mises au point par Pierre Bézier, ingénieur chez Renault, pour représenter les formes de pièces mécaniques.

(2) Tous les schémas de ce livre ont été écrits en PS par l'auteur.

On trouvera ci-dessous un programme PostScript provoquant le tracé de la figure en regard. Les deux premières lignes constituent l'en-tête normalisée. Les commentaires sont précédés d'un signe %. Les instructions précédées du signe / sont des définitions de procédures ou de termes. Cet exemple prouvera au lecteur que, si ce langage n'a pas la simplicité du Basic, il est rationnel et parfaitement compréhensible.

```
%!PS-Adobe-2.0
%%BoundingBox : 0 0 100 100
/trl {translate}  def          % translation de x et y
/mv {moveto} def               % aller au point x, y
/rm {rmoveto} def              % se déplacer de dx et dy
/rl {rlineto} def              % tracer une ligne sur dx et dy
/ferm {closepath} def          % fermer le tracé en cours
/deb {gsave} def               % nouvel état graphique
/ret {grestore} def            % retour à l'état précédent
/term  {deb setgray fill ret stroke}  def     % fin d'une face
/point {mv currentpoint r 0 rm r 0 360 arc g term} def
                % point
% après les définitions, voici le tracé
deb 2.8345 dup scale           % coordonnées en mm
5 mm 5 mm scale   1  1  trl    1 setlinejoin    0.0 setlinewidth
/a 1.8  def        /r 0.4  def
0 0 mv  4 0 rl  0 4 rl  -4 0 rl ferm  0.97  term          % face 5
4 0 mv  a a rl  0 4 rl  a neg a neg rl ferm .85 term  % face 2
0 4 mv  4 0 rl  a a rl -3.5 0 rl ferm  0.6 term          % face 1
                % puis les points sur chaque face
/g .0 def  1 1 1 3 2 2 3 1 3 3 5 {point} repeat        % face 5
/g .4 def  deb 4 0 trl a 4 div 1 scale 1.5 1.8 3 4.2  2 {point}
repeat  ret   % points face2
deb 0 4 trl 1 a 4 div scale
3.1 2.0 point  ret    % face 1
/Times-Roman  findfont  0.4
scalefont  setfont
% police & corps
deb  4.7 0 mv 49 rotate
(alea jacta est) show  ret
% texte

showpage        % ordre d'imprimer
```



2 - UTILISATION DE LOGICIELS DE DESSIN

Il existe des logiciels commercialisés facilitant considérablement la mise au point des dessins. Appelés parfois *grapheurs*, ces logiciels ont un nom comprenant souvent les termes *paint* ou *draw*. Ils produisent des dessins soit du type **image de points** *(bitmap*, type *paint)*, ceux créés pour l'écran, soit du type **vectoriel** *(draw)*, destinés à des traceurs, c.-à-d. presque à coup sûr, écrits en HPGL ou en PostScript.

Ces logiciels affichent à l'écran une sorte de *boîte d'outils*, regroupant en bordure tous les *objets* disponibles. Parmi eux, on retrouve ceux auxquels nous sommes habitués : droite, rectangle, cercle, parfois ellipse. Y figurent aussi la ligne brisée et des courbes, mais pas toujours celles, très artistiques, de Bézier.

Il faudra commencer par définir ici encore un *état graphique* : feuille de papier, largeur du trait, style (continu, pointillé), couleur, échelle des coordonnées.... Puis, grâce à la méthode habituelle (souris, touches flèches, RC), on sélectionne l'un des objets disponibles. On amène ensuite cet objet à sa place et on lui donne l'orientation et la taille voulues. Petit à petit, on construit un dessin complet à partir des objets élémentaires affichés. Tout cela avec l'assistance de menus relativement conviviaux.

D'autres outils sont disponibles, comme la *gomme* ou tout autre moyen capable d'effacer. L'outil le plus puissant est certainement celui qui permet de **grouper** plusieurs objets en un seul nouvel objet, qui pourra parfois être manipulé de la même façon que les premiers : déplacé, agrandi, pivoté, symétrisé, regroupé, etc. Une autre facilité offerte consiste dans le quadrillage *magnétique*, qui peut n'avoir qu'un rôle indicatif. Mais, sur demande, il se dote d'un pouvoir attractif tel que les objets proches soient automatiquement amenés au contact d'un nœud du réseau. Cela restreint la liberté, mais améliore beaucoup les alignements.

Ces logiciels sont d'un intérêt inégal. Certains produisent des schémas techniques ou scientifiques de qualité moyenne, ou des images aux formes géométriques. Il doivent disposer de fonctions indispensables : traits d'épaisseur variable, faculté de modifier tout objet (taille, orientation, couleur ...), sélection facile d'un objet déjà placé, précision dans le groupage-dégroupage. Bien peu assurent toutes ces fonctions et beaucoup ne sont guère plus utiles qu'un logiciel de jeu. C'est pourquoi l'on devra souvent se rabattre sur les méthodes de programmation *pures et dures* étudiées précédemment, à moins de pouvoir s'offrir l'un des logiciels de haut niveau étudiés ci-après.

3 - LA CAO

Les logiciels dont on parlera ici ont été développés pour réaliser du *dessin industriel*. Leur exploitation est désignée sous le nom de **conception assistée par ordinateur**. Toutes les possibilités évoquées plus haut leur sont dévolues, avec en plus certaines autres comme la cotation automatique, le travail sur plusieurs couches, la notion de troisième dimension... Ils sont caractérisés par une précision maximum en ce qui concerne le positionnement des **objets** : les coordonnées exactes des points désirés sont affichées à l'écran et on peut mettre en place un objet avec une commande frappée au clavier et non plus seulement avec la souris, trop imprécise pour du dessin vectoriel.

Parmi les outils non encore cités, mentionnons un jeu complet de *hachures* (pour *habiller* les figures), la disposition de coordonnées polaires souvent bien utiles et la fonction d'*accrochage*, qui permet d'ancrer une nouvelle ligne en un point précis d'une figure déjà constituée. L'exécution d'un même dessin en plusieurs couches superposables permet de gagner beaucoup de temps (par exemple, dans le cas d'un plan de bâtiment, on fera une couche par corps de métier).

Ces logiciels peuvent parfois fournir des *perspectives cavalières (vues d'artiste)*, habillées et coloriées comme décrit §7. Ainsi, grâce à la rotation programmée du repère de coordonnées, un architecte peut présenter à son client un film de sa future maison, vue comme si on tournait autour d'elle en hélicoptère !

Un des très gros intérêts de la CAO est la constitution de bibliothèques d'*objets* dessinés une fois pour toutes et insérables dans tout nouveau dessin.

L'aboutissement normal d'un tel travail est le tracé comme décrit § 3 (p. 119), mais sur *traceur* de grande taille. On peut obtenir des *plans* en format A2, A1, A0. Le langage intermédiaire entre traceur et logiciel est souvent le HPGL. Mais ces logiciels sont parfois capables de *sortir* leur produit sous plusieurs langages, dont PostScript. Il est possible d'obtenir un dessin de taille modeste, ou une réduction, sur imprimante laser, si elle est munie d'un interpréteur pour le langage vectoriel employé (PostScript ou HPGL).

Passés les délais nécessaires à leur apprentissage, le travail réalisé avec les logiciels de CAO est excellent et le gain de temps considérable. Ils sont rarement d'un emploi facile, ne brillent pas par leur convivialité et coûtent cher. Aussi ne sont-ils guère accessibles – pour l'instant du moins – qu'aux professionnels des *bureaux d'études*.

4 - LE GRAPHISME ARTISTIQUE

Pour obtenir un dessin plus expressif, plus libre ou plus réaliste, c.-à-d. plus proche de la photo ou de celui de l'artiste, il faut plus de moyens. L'évolution en ce domaine est permanente, aussi serons-nous loin d'être exhaustifs dans l'énoncé des procédés offerts.

Pour se rapprocher de la peinture, la **couleur** doit pouvoir varier de façon continue. Les écrans en mode VGA autorisent normalement 16 couleurs, ce qui est suffisant pour le dessin technique, mais pas pour le dessin d'art. Le matériel graphique a beaucoup progressé depuis peu et on peut disposer maintenant d'un très grand nombre de couleurs (jusqu'à 2^{24}, plus de 4 millions) ou de plus du million de points sur l'écran (mais rarement simultanément, voir section 8).

Un autre point important pour le dessin *figuratif* (i.e. tel que l'œil voit la scène) consiste dans ce qu'on appelle le *rendu* des **parties cachées**. En programmation élémentaire, ce résultat est difficile à obtenir (sauf en PostScript). On ne peut plus travailler avec des figures toutes prêtes (rectangles, ellipses, etc). Il faut calculer pour chaque point, en plus de ses coordonnées, une variable logique définissant s'il est visible ou caché. Cette caractéristique est définie par rapport à un *masque*, figure générale regroupant toutes celles situées dans un *plan* supérieur au plan utilisé. Il faut donc programmer le dessin de plan en plan, en commençant par le plus proche. Les logiciels d'assistance offrent tous des possibilités de rendu des parties cachées, et ceci quel que soit l'ordre du tracé des plans. Cependant, le procédé échoue parfois pour quelques formes, les objets concaves en particulier, dont certains logiciels parviennent mal à distinguer l'intérieur de l'extérieur.

Il faudra donner ensuite à la scène une impression de **relief** (ou *de troisième dimension*). Les logiciels de haut niveau le peuvent et spécialement ceux de CAO. On peut aussi l'obtenir par programmation en ajoutant des lignes dont l'orientation est calculable (elles convergent vers le *point de fuite*) et la longueur égale à celle de leur *projection*, après affectation d'un coefficient de profondeur. Ce n'est déjà pas tellement simple et, pourtant, ce n'est pas fini ...

Jusqu'ici, on ne s'est guère intéressé qu'au *contour* des objets. La représentation de **surfaces courbes** exige la maîtrise de techniques nouvelles. Il faudra d'abord construire un **maillage** régulier de points sur la surface et en représenter la projection sur le plan du dessin. On reliera ces points par des segments de droite **(3)**. Plus la maille sera serrée, plus le réalisme sera approché. Ces segments délimitent des figures simples et planes, triangles ou trapèzes que l'on va colorier ou recouvrir d'un motif (parfois improprement appelé *texture*) de façon à plaquer en quelque sorte une *peau* sur l'objet. Une autre méthode consiste à faire glisser une courbe, appelée *section*, le long d'une autre appelée *élévation*, en la modifiant bien sûr à chaque position. Par exemple, l'aspect d'un tore (anneau creux) est obtenu par glissement d'une ellipse de rapport d'aspect variable, mais de grand axe constant, le long d'une autre ellipse plus grande.

Le summum de la technique figurative est atteint par les **effets d'éclairage**, c'est-à-dire par le rendu du jeu subtil entre la lumière ambiante et les différentes parties d'un objet. Un objet donné, même de couleur uniforme, présente des ombres et des parties claires selon son exposition à la lumière. Deux facteurs entrent en jeu pour déterminer la *brillance* locale de la surface d'un objet : son "pouvoir réflecteur" (qu'on

(3) Un tel maillage est utilisé par les ingénieurs pour décomposer une pièce, un ouvrage d'art... en *éléments finis* et calculer de proche en proche les efforts qu'il subit. Il est disponible en CAO et s'appelle alors souvent *modélisation*.

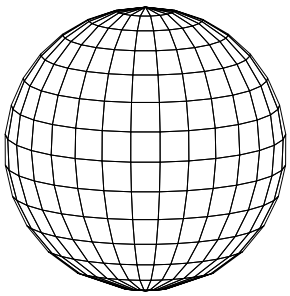
devrait plutôt appeler *albédo* ou pouvoir diffuseur) et l'angle des rayons lumineux l'atteignant. Un logiciel prétendant rendre les effets de lumière devra donc calculer, pour de nombreux éléments de surface, le **tracé des rayons** lumineux l'éclairant (*ray-tracing*), l'angle d'incidence et celui de diffusion vers le spectateur, puis jongler avec l'albédo de la surface et les cosinus de ces angles pour calculer le pourcentage de clarté sur la surface étudiée. On comprend aisément que cette technique soit extrêmement laborieuse à programmer. Cependant, elle figure parmi les possibilités des logiciels de dessins haut de gamme.

Pour terminer cette section, revenons sur les techniques d'**animation**. On a vu que le Basic offrait déjà des moyens non négligeables dans ce sens. Il s'agissait pourtant d'images simples, sans problème de relief, de parties cachées, de peau, ou de calcul de rayons. Quand les dessins dotés de tels raffinements exigent en plus le mouvement, on se trouve confronté à des problèmes difficilement solubles, les images devant souvent, comme au cinéma, défiler au taux de 25 au moins par seconde. La *puissance de calcul* nécessaire dépasse alors celle des ordinateurs personnels, même avec *coprocesseur*. On doit faire appel à des calculateurs rapides ou bien à des *cartes d'extension* optimisées pour le dessin. Il faudra bien se garder de modifier à chaque vue la totalité de l'image affichée et se contenter du strict nécessaire, par exemple en *comprimant* les images (voir section suivante).

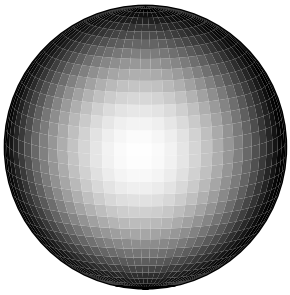
Il faut signaler que ce genre de dessin n'intéresse pas que les artistes, mais également, entre autres, les techniciens de la **simulation**. Un *simulateur de vol* est un logiciel dessinant à l'écran un paysage tel que le verrait un pilote d'avion selon la conduite qu'il imprime à son (pseudo-)avion. Tout ce paysage est modélisé et chacun des éléments fait l'objet de calculs poussés et rapides. De même, les logiciels de **jeux informatiques** sont de gros consommateurs de dessin animé, mais avec des exigences bien moindres.

Les figures suivantes feront saisir au lecteur la quantité de calcul nécessaire à la création de dessins élaborés, même si le travail impliqué reste invisible à l'utilisateur d'un logiciel d'aide à la création graphique. Ce sont les auteurs du logiciel qui ont fait les calculs pour lui.

La première figure montre un maillage sur une forme simple, la sphère. Le maillage (ou structure en fil de fer*) consiste à réunir par des segments de droite les points obtenus par calcul des projections. Ici, chaque point est l'intersection de deux familles d'ellipses, projections elles-mêmes des méridiens et parallèles de la sphère.*



La seconde figure dérive de la première avec un maillage plus serré (35 ellipses au lieu de 15). Ensuite, chacun des trapèzes obtenus par maillage a été "coloré" en gris avec une intensité proportionnelle à la lumière diffusée pour un éclairage censé provenir de la direction d'observation.



5 - LES IMAGES ECHANTILLONNEES

Une image échantillonnée, ou plus simplement une **image**, est une représentation graphique décrite point par point *(bitmap)*. Beaucoup de logiciels graphiques créent de telles images, qui sont de type *paint*. Elles peuvent être également saisies par un *scanner* à partir de photos ou de dessins manuels. Un *scanner* (le terme correspondant français serait sans doute *bélinographe*) balaye une image ligne par ligne en traduisant par une information élémentaire la couleur ou la luminosité de chaque point. Le point ultime ainsi traduit s'appelle *pixel* ou *échantillon* ; c'est une petite surface dont l'aire dépend de la résolution du scanner, i.e. du nombre de capteurs n_x qu'il possède sur sa ligne de saisie.

L'image comprendra donc $n_x\, n_y$ pixels si elle est traduite par n_y lignes. Si l'image initiale consiste en un *dessin au trait*, on peut se contenter de saisir seulement le noir et le blanc : aucune nuance ne sera autorisée. Alors chaque pixel se traduira par un bit égal à zéro si le pixel était noir et à un si le pixel était blanc. Le volume total du fichier représentant l'image est de $n_x n_y /8$ octets. Si l'image est nuancée comme dans le cas d'une photo, on doit enregistrer le niveau de gris de chaque pixel. On le fait souvent en distinguant 256 niveaux[(4)] ou 8 bits. Si on veut enregistrer la couleur de l'image, il faudra saisir 3 informations de ce type, une par couleur fondamentale. La taille du fichier devient alors $3n_x n_y$ octets. C'est considérable. Une image de télévision "noir et blanc" par exemple comporte $625^2 \times 4/3$ pixels, plus d'un demi-million. En couleur avec 256 niveaux par couleur (3×8 bits par pixel), elle occuperait plus de 1,5 mégaoctet.

Pour conserver ces images, il est presque indispensable de les comprimer (*compress* en anglais). Plusieurs types de compression sont utilisés. Le plus simple (procédé **RLE**) remplace par deux octets de valeur *n* et *k* toutes les séquences formées de

(4) Parce que c'est plus pratique pour l'informatique. Mais en général 64 niveaux suffiraient pour l'œil humain moyen.

n octets successifs égaux à *k*. Ce procédé comprime bien les dessins au trait, car les fractions de lignes blanches y sont importantes. Il fonctionne mal avec les autres images et d'autres procédés plus complexes sont alors employés. On distingue deux grands types de procédés de compression, ceux qui conservent intégralement l'information et ceux qui, jouant sur la résolution finie de l'œil, acceptent de perdre une partie de l'information comme celui mettant en œuvre la transformation de Fourier, dont on tronquera le spectre du côté des hautes fréquences (perdant ainsi les détails les plus fins).

L'image la plus à droite a été saisie par scanner sur un dessin à l'encre de chine, donc au trait (5) *; avec 570 lignes de 304 pixels et 1 bit par pixel, elle occupe 21 660 octets. La photo de gauche, avec ses 210 lignes de 178 pixels et 8 bits par pixel occupe 37 380 octets.*

6 - TRAITEMENT DES IMAGES

Les images contenues dans des fichiers informatiques sont toujours *sérialisées* : toutes les valeurs relatives aux pixels se suivent. Pour reconstituer l'image, il faut connaître n_y, nombre de lignes, n_x, celui de pixels par ligne et n_b, celui de bits par pixels. Ces précieuses informations sont jointes au fichier image selon des conventions propres à chaque type d'image. Ce type se reconnaît grâce au suffixe du fichier ; ainsi, des fichiers suffixés PIC, GIF, ... signalent qu'ils contiennent des images mémorisées selon un format bien défini.

Le format probablement le plus universel est le format TIF. C'est aussi le plus difficile à relire ; mais, simple à mettre en œuvre en phase saisie, il est très employé par les scanners et donc très courant.

Une fois la structure de l'image bien comprise, on peut la modifier, effectuer des coupures, des changements de teintes... Plus difficiles sont les réductions ou agrandissements, une image échantillonnée présentant une certaine rigidité vis-à-vis de ces modifications d'échelle (alors qu'un dessin vectoriel ne poserait aucun problème). La manipulation la plus spectaculaire consiste à remplacer une partie d'image par une autre, par exemple à *truquer* une photo en rapprochant deux personnages pour faire croire à leur connivence. C'est un procédé lourd de conséquences, car jusqu'ici, la photo possédait une certaine force de preuve ; cette crédibilité dans l'image devra disparaître devant la facilité de manipuler les images informatiques.

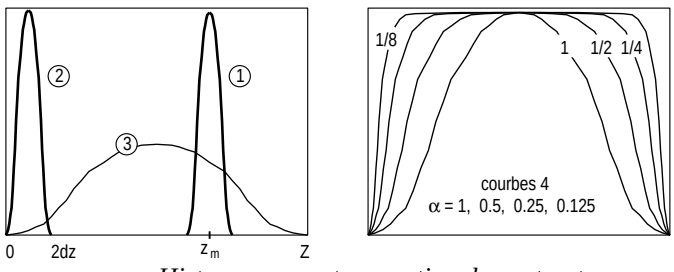
Un traitement plus innocent consiste à améliorer l'aspect (contraste, piqué, ...) d'une photo. On peut procéder par essais avec un logiciel de retouche d'image. Il est plus rationnel de commencer par tracer ce qu'on appelle l'**histogramme** de l'image ; c'est une courbe donnant la fréquence statistique du niveau de gris ou de la teinte de ses pixels.

Le pixel étant représenté sur Z niveaux (très souvent 256), l'histogramme donne le nombre de pixels ayant la valeur z ($0<z<Z$). Si l'histogramme est bien étalé, la photo ou l'image est bonne ; s'il est étroit, on pourra améliorer le rendu de l'image.

Par exemple, si l'histogramme ressemble à un pic centré en z_m de largeur $2dz$ (courbe 1), on pourra amener ce pic près du zéro en ôtant $z_m - dz$ à tous les z (courbe 2), puis les multiplier par $Z/(2dz)$ pour étaler l'histogramme sur toute l'étendue des z (courbe 3). L'image sera beaucoup plus nuancée, mais restera douce d'aspect. Pour durcir les contrastes, il faut donner à l'histogramme une allure *plus carrée*. On peut remplacer les valeurs corrigées de z (courbe 3) par la fonction $\frac{1}{2} Z\{1 + [(2z/Z)-1]^{\alpha}\}$ avec $\alpha = 1, 0.5, 0.25...$ (courbes 4).

Certains logiciels autorisent ces opérations précises et reproductibles. En leur absence, on peut encore faire appel à PostScript, qui permet d'imprimer facilement une image de points (avec d'ailleurs n'importe quel facteur d'échelle) ; sa *fonction de transfert* u ($u = z/Z$, $0 \le u \le 1$) se modifie aisément par l'instruction suivante, où les opérations entre les accolades font correspondre une nouvelle valeur de u à l'ancienne (avec $um = z_m/Z$ et $du = dz/Z$) :

```
/currenttranfer {um sub du add Z mul 2 du mul div}
settransfer
```



Histogrammes et correction de contraste.

(5) "Dauphinoise", dessin de Lisette Blanc.

7 - AFFICHAGE DES IMAGES

Dans la section 2, on a appris à représenter à l'écran des lignes, droites ou courbes. En effet, sur les écrans anciens, le point-écran ne peut être qu'éteint ou allumé. Pour simuler les nuances, il faut représenter un pixel non par un point-écran, mais par un ensemble de points (procédé appelé *matriçage*). Ainsi, on peut décider de représenter un pixel-image par 4×4 points écran, ce qui autorise 17 nuances, si on les simule par l'allumage progressif des 16 points (il faut faire attention à l'ordre d'allumage, sinon on créera des stries ou des moirés dans l'image représentée). La résolution spatiale pâtit de ce procédé. Sur un écran EGA (640×350 points), on ne peut représenter ainsi que 160×87 pixels en 17 nuances ou 128×70 (26 nuances).

Avec les écrans *analogiques* (VGA, SVGA...), l'intensité d'un point est modulée pour représenter niveau de gris ou intensité de couleur. Il est alors possible de faire correspondre point-écran et pixel-image, ce qui améliore fortement la résolution spatiale. Il faudra beaucoup de mémoire pour contenir de telles images. Ainsi, la carte VGA ne disposant que de 150 ko ne peut fournir que 16 teintes en haute résolution (640×480). Or elle occupe tout l'espace d'adressage autorisé pour l'affichage dans un PC. Pour avoir plus de résolution ou plus de teintes, on a recours au procédé SuperVGA : il exige la présence d'une quantité de mémoire suffisante sur la carte vidéo. Par exemple, 1 Mo autorise 256 teintes ou couleurs pour 1024×768 pixels ou 65536 couleurs pour 800×600 pixels, soit $n_{\mathbf{x}} n_{\mathbf{y}} \log_2 N_{\mathbf{t}} / 8$ octets de mémoire si $n_{\mathbf{x}}$, $n_{\mathbf{y}}$, $N_{\mathbf{t}}$ sont égaux aux nombres de lignes, de pixels par ligne et de teintes. La mémoire de la carte SVGA n'est pas directement adressable ; elle communique avec l'UC par pages de 64 ko (sa programmation est donc difficile).

8 - IMPRESSION DES IMAGES

On a déjà vu (note 5, p. 118) qu'on pouvait, sous DOS, imprimer des images-écran. On peut également les enregistrer dans un fichier imprimable avec l'aide d'un utilitaire, souvent appelé **CAPTURE**. Mais ces images gardent la faible résolution de l'écran initial.

L'impression directe de belles images est aussi difficile que leur présentation sur un écran EGA, car les imprimantes (sauf celles à sublimation, à vrai dire encore rares) ne savent déposer que du noir sur du blanc. C'est un procédé binaire. Pour simuler le gris, on employait autrefois des hachures d'espacement variable. Pour imprimer des photos, on utilise le *tramage*. Ce procédé consiste à échantillonner l'image en la réduisant à un réseau de taches équidistantes (donc situées sur un quadrillage régulier), mais de grandeur variable. Chaque tache sera en gros circulaire avec un rayon proportionnel au niveau de gris de la photo en cet endroit. La couleur est rendue par juxtaposition de trois réseaux tramés encrés avec l'une des couleurs primaires. On peut obtenir le tramage par des procédés optiques (diffusion de la lumière à travers une grille), mais en impression informatique, le tramage est obtenu de façon purement logicielle, après saisie de la photo par scanner.

La tache qui représente le *n*ième niveau de gris s'obtient par agglutination des *n* points-imprimante les plus proches de son centre. On appelle *linéature* la densité des taches (son inverse est *p*, l'intervalle entre lignes). Si la distance entre points-imprimante est *r* (son inverse est la résolution), le nombre de niveaux de gris possibles est $1 + (p/r)^2$, car la tache contient au maximum $(p/r)^2$ points-imprimante.

Ainsi, une imprimante de "résolution" 300 *dpi* (points par pouce) ne peut reproduire que 17 niveaux de gris si on exige une linéature de 75 lignes par pouce *(lpi)* ; c'est à peine égal à la qualité des photos dans les quotidiens. Une imprimante de 600 *dpi* fournira, toujours avec 75 *lpi*, 65 niveaux de gris, ce qui est suffisant puisque l'œil humain ne distinguerait pas davantage de teintes, mais la linéature est trop grossière. Pour égaler la qualité d'une photo dans les hebdomadaires, il faut une linéature de 150 lignes et 65 niveaux, soit une résolution de 1200 dpi. C'est une résolution encore réservée au domaine professionnel.

Là encore PostScript est remarquable dans l'impression des photos ou des images échantillonnées *(bitmap)*, car il gère lui-même le redimensionnement et le tramage. Il permet d'atteindre le maximum de ce qu'autorise la mécanique de l'imprimante. Voici, pour terminer, une photo avec des détails progressifs qui feront comprendre le mécanisme du tramage et la façon d'imprimer des taches de diamètre variable avec des points de largeur fixe.

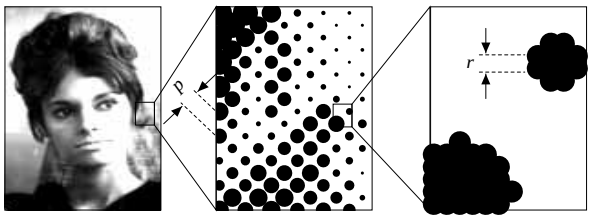


Photo tramée *linéature (tramage)* *résolution (taches)*

Chapitre XV

LA TELEMATIQUE

On propose de définir la *télématique* [1] comme le traitement de l'information à distance. Le sujet étant trop vaste, nous n'en retiendrons que l'aspect **communication**. Nous commencerons par en étudier la partie concrète, c.-à-d. les *supports* les plus utilisés pour ce genre de communication : fils électriques, paires téléphoniques, câbles coaxiaux, fibres optiques, ondes électromagnétiques.

L'examen des méthodes permettant le transfert de l'information nous amènera à distinguer les liaisons parallèles des liaisons série. On donnera trois exemples de **liaisons parallèles** très importantes pour les ordinateurs personnels, la liaison Centronix pour les imprimantes, les bus *externes* et le bus GPIB.

Nous pénétrerons ensuite dans le vaste domaine des **communications série**, en partant de celles vouées aux distances les plus courtes (RS232) pour survoler celles opérant à moyenne distance , les *réseaux locaux (LAN, local area network)* et terminer avec les *grands réseaux (WAN, wide area network)*, plus spécialement avec le réseau mondial **Internet** et les services qu'il offre au grand public.

1 - LES SUPPORTS DE COMMUNICATION

Le plus connu de ces supports est le fil électrique. Le *télégraphe* l'a adopté à ses débuts. Les tentatives d'alors pour utiliser un seul conducteur en refermant le circuit *par la terre* ont vite révélé les inconvénients du procédé : perturbations des communications par les courants telluriques et par les champs électromagnétiques ambiants, qui introduisent des *parasites* ou du *bruit*. On a donc dû utiliser des **paires de conducteurs**, d'abord parallèles, puis torsadés, beaucoup moins perturbés par les champs extérieurs [2].

La **vitesse de transmission** de l'information sur ces lignes est presque égale à *c*, celle de la lumière dans le vide, le rapport entre les deux dépendant de l'isolant entre les conducteurs. Bien que très élevée (3.10^8 m/s, soit un délai de 3,3 µs/km), la valeur *finie* de *c* est source de difficultés dans les communications lointaines ou pour celles de débit élevé.

Il faut bien faire la différence entre *vitesse de transmission* et *débit de ligne*, hélas souvent appelé *vitesse de communication*. Cette dernière caractéristique très importante s'exprime en bits par seconde ou bps [3]. Pour saisir la nuance, pensons au guetteur indien des Western envoyant un message avec des signaux de fumée : bien que ce message parvienne au destinataire à la vitesse de la lumière, la quantité d'information transmise par minute est très faible (quelques bits). Le télégraphe optique de Chappe n'avait pas lui non plus un fort débit.

Deux propriétés de la ligne conditionnent son débit N' : sa *modulabilité*, vitesse de variation du signal (cf. chap. III, §5, p. 17) et sa *dynamique*. La modulabilité est nécessairement inférieure à sa *bande passante F*, laquelle est égale à la fréquence maximum du signal pouvant transiter par la ligne. Si le message informatique est formé de bits, ils sont représentés par des impulsions de courant qui ne peuvent être plus brèves qu'environ deux fois leur temps de montée τ (cf. fig. 3-6, courbe *b*, p. 17). Le débit maximum de la ligne est donc dans ce cas $N' = 1/2\tau$. Or, selon Fourier, un tel signal est assimilable à une somme d'ondes sinusoïdales dont la fréquence maximum f est liée à la durée τ du front de montée par la relation $f \approx 1/\pi\tau$, ce qui entraîne $N' \approx \pi F/2$ $(\approx 1{,}57F)$ [4], la valeur de f étant au plus égale à la *bande passante F*.

L'autre caractéristique majeure d'une ligne est ce qu'on peut appeler sa *dynamique*. Si l'on se contentait d'envoyer des signaux *tout ou rien* (comme le guetteur indien ou avec la liaison RS232, cf. §4), on ignorerait cette propriété fondamentale d'une ligne électrique qui est de pouvoir acheminer des tensions ou des courants d'intensité variable. Cette propriété est peu utilisée dans le téléphone, car la compréhension

---

(1) Terme créé par S. Nora et A. Minc dans *L'informatique de la société* (La documentation française, 1978).

(2) Le bruit étant lié au *flux* du champ parasite à travers la surface du circuit, on *torsade* les deux fils, ce qui réduit son aire et minimise le flux algébrique grâce à la rotation continue du vecteur caractérisant cette surface (sa normale orientée).

(3) Autrefois, l'unité en était le baud. Ce nom paraît maintenant réservé à la *modulabilité*. Autre illogisme : *bps* (anglicisme) qui devrait s'écrire *b/s* .

(4) Pour retrouver la valeur donnée par le théorème de Shannon (cf. p. 128), il suffit de remarquer qu'on a affaire à deux niveaux seulement. Avec $S=2B$, $N' = F \log_2(1+S/B) \approx 1{,}58F$.

de la voix est plus liée à la hauteur des sons qu'à leur *force* (leur intensité), pourvu qu'ils émergent du bruit.

Le long des lignes, les signaux s'*atténuent*. Au bout d'une certaine longueur, il faut placer un *répéteur* pour les réamplifier (la contrainte imposée par ces répéteurs grève lourdement le budget de la ligne, surtout si elle est sous-marine). Même avec ces dispositifs, l'*émetteur* n'est pas maître du niveau reçu par son correspondant. C'est pourquoi le codage sur plusieurs niveaux a longtemps été négligé. Shannon avait pourtant montré que le débit maximum d'information sur une ligne était en théorie égal à $N' = F \log_2(1+S/B)$, si F est sa bande passante et S/B son rapport signal sur bruit (ou sa dynamique), i.e. le rapport entre le plus fort signal admissible et le bruit. Pour s'approcher de cette limite, il faut utiliser des tensions variables, malgré les atténuations et amplifications inconnues interposées entre les deux interlocuteurs.

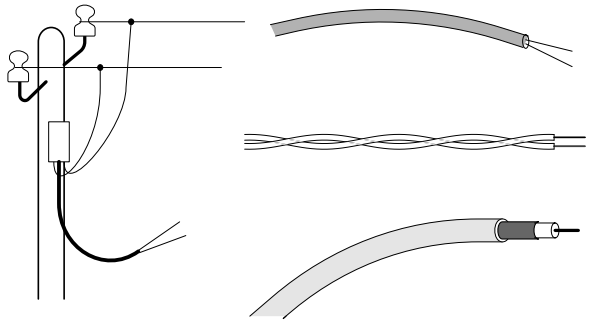
On n'a pu y parvenir qu'avec les **filtres adaptatifs**. Cette technique consiste à envoyer de temps en temps sur la ligne un signal de référence dont le récepteur mesure le niveau pour déterminer l'atténuation de la ligne. Les niveaux des signaux ultérieurs seront alors significatifs, dans la limite de Shannon.

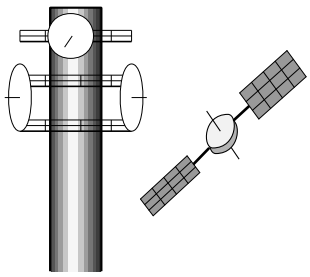
Bande passante et atténuation dépendent des mêmes phénomènes physiques : perte de l'énergie électrique par rayonnement d'une part et par échauffement des conducteurs et isolants d'autre part. C'est donc ces points qu'il faut améliorer pour obtenir des lignes de qualité. Ces améliorations coûtent cher ; aussi, trouve-t-on un large éventail de supports de communication, selon l'importance de la liaison envisagée.

Les *paires électriques* exigent des répéteurs tous les 10-15 km ; leur bande passante est faible : 3,4 khz par exemple pour le réseau téléphonique de base. C'est une bande bien modeste pour un échange informatique. On peut cependant atteindre quelques Mhz sur de courtes distances avec une paire munie d'un blindage adéquat. Cette valeur est même dépassée sur les lignes ou torons constitués de plusieurs *paires*. A ce stade, la ligne entre en compétition avec le *câble coaxial*, dont la bande passante peut dépasser 100 Mhz avec une atténuation faible, mais au prix d'une construction et d'une connexion plus délicates.

Des débits encore supérieurs s'obtiennent avec les **faisceaux** d'ondes électromagnétiques déci- ou centimétriques qui se propagent en ligne droite de relais en relais. Ces relais sont des émetteurs-récepteurs radio placés sur des hauteurs distantes d'environ 100 km (souvent celles où Chappe avait fait installer ses sémaphores). Le **satellite** artificiel est un cas particulier de relais en ondes centimétriques qui, situé à grande hauteur, est *vu* par une fraction notable de la planète (près de la moitié de la Terre s'il est *géostationnaire*) [5]. Enfin, depuis environ 20 ans, on sait fabriquer des **fibres optiques** dont la bande dépasse un Ghz ; leur qualité s'améliore d'année en année. On peut construire des lignes optiques ne nécessitant de répéteurs que tous les 100 km environ. Leur prix est cependant plus élevé que celui du câble, lui même supérieur à celui d'un toron de paires.

Tous ces développements sont liés à celui du téléphone, qui ne nécessite pourtant qu'une bande de 3,4 khz pour transmettre une conversation. On a cependant recherché des bandes élevées parce qu'on sait **transposer les fréquences**. Par exemple, on peut acheminer simultanément deux conversations téléphoniques sur un même câble de bande passante 7,4 khz, la première occupant la bande 0–3700 hz, la seconde, transposée, la bande 3700–7400. Ainsi, un seul câble coaxial de bande 50 Mhz est capable de véhiculer jusqu'à $5.10^7/3700$, soit environ 13500 conversations **simultanées**, et beaucoup plus dans le cas d'une fibre optique. Le jeu en vaut la chandelle. C'est cette technique de *multiplexage* qui a fait disparaître les nappes de fils téléphoniques longeant autrefois nos voies ferrées et les a fait remplacer par un petit nombre de câbles coaxiaux ou de fibres optiques enterrés le long des routes ou bien par des *artères* de faisceaux radio.





Les supports de communication : *on reconnaît dans ces schémas la paire téléphonique ordinaire (en haut à gauche), la double fibre optique (en haut à droite) et, en descendant, la paire torsadée, puis le câble coaxial. Sont symbolisés ci-contre, un relais de faisceaux radio à paraboles et un satellite artificiel.*

(5) Un satellite est *géostationnaire* s'il est fixe par rapport à la Terre, donc s'il tourne à la même vitesse angulaire ($\omega = 2\pi/24$ rad/heure), ce qui implique une orbite de rayon $r \approx 36000$ km et une vitesse orbitale voisine de $r\omega \approx 3000\pi \approx 9425$ km/h.

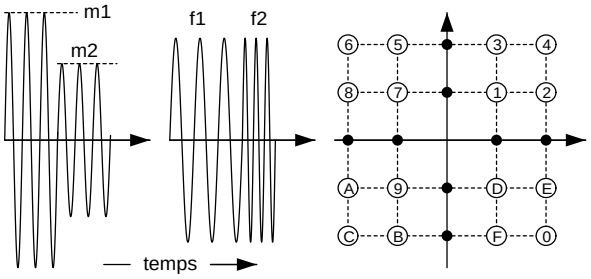
2 LES MODEMS

On verra que les transmissions informatiques sont souvent obligées d'emprunter le réseau téléphonique classique ou *commuté* (RTC). Or ce réseau est optimisé pour transmettre des signaux sinusoïdaux (cf. chap. III, p. 17) produits par des *micros* recevant des sons. Il réagit très mal avec les signaux *tout ou rien* de l'informatique. A cause de cela, on passe par un *modem* (*modulateur-démodulateur*), petit appareil capable de *moduler* par le signal informatique une onde de fréquence fixe, la *porteuse*. Les modems peuvent moduler en amplitude, fréquence ou phase, une ou parfois plusieurs porteuses.

Leur débit est passé progressivement de 300 bits par seconde à 14400 sur le RTC, grâce à la modulation d'amplitude avec filtre adaptatif. Le procédé actuellement le plus employé sur le RTC est le MAQ (modulation d'amplitude de deux porteuses en quadrature de phase et d'égale fréquence). Si la modulation repose sur m niveaux pour chacune des deux porteuses, on obtient m^2 états ou *codes* possibles, qui apportent donc autant d'information que $q = \log_2 m^2$ bits. Si la rapidité de modulation est B (en bauds), le débit d'information N' en b/s est égal à qB.

$$N' = 2B \log_2 m$$

De cette façon, on peut transmettre 9600 bps sur 2 porteuses de fréquence 2400 hz (B = 2400 bauds) avec 4 niveaux d'amplitude par porteuse (soit 16 états, q=4) ; c'est la *recommandation* (*avis*) V32 du CCITT. Avec les mêmes porteuses et 8 niveaux, on obtient un débit de 14400 bps (64 états, q=6, norme V32bis). On remarquera qu'avec la norme V32, on transmet juste un quartet (16 états) par baud.



A droite, modulation en amplitude sur 2 *niveaux (*m1 *et* m2*). Au centre, modulation en fréquence sur* 2 *valeurs* f1 *et* f2 *. A droite, diagramme des* 16 *états (quartet* 0..F*) dans une modulation type* MAQ *à* 4 *niveaux sur* 2 *porteuses en quadrature.*

Ce type de transmission, très répandu, exige un modem à chaque extrémité de la liaison. C'est lui qui, à la réception, retransforme les ondes sinusoïdales en signaux binaires, seuls compréhensibles par les calculateurs. Les modems récents savent modifier leur débit nominal et passer automatiquement à un débit inférieur si la qualité de la ligne l'exige.

Par exemple, le réseau informatique *Minitel* utilise un modem pour transmettre ses bits sur les lignes téléphoniques. Les conventions sont les suivantes : sur la *voie réception*, les bits 1 et 0 sont représentés par un *train d'ondes* aux fréquences respectives de 1300 et 2100 hz, et sur la *voie émission*, par des trains à 390 et 440 hz. Les débits moyens pour chaque voie sont respectivement de 1200 et 75 bauds. Il est possible d'intervertir le rôle des deux voies pour des applications particulières, lorsque le modem est de type *retournable*.

Les *téléphonistes* ont donc obligé les informaticiens à convertir dans des modems leurs signaux binaires en courants sinusoïdaux. Juste retour des choses, les ingénieurs du téléphone ont vite été séduits par la supériorité du numérique, à tel point qu'ils viennent progressivement au binaire, allant jusqu'à *numériser* la voix pour la coder en une suite de bits. Les signaux binaires sont en effet beaucoup plus faciles à réamplifier et à protéger du *bruit* que les signaux sinusoïdaux.

Ces techniques sont en service sur les liaisons à grande distance, spécialement sur celles utilisant des *satellites*. C'est grâce à elles qu'on obtient souvent avec un correspondant étranger une conversation téléphonique de bien meilleure qualité qu'avec son cousin de banlieue. La généralisation de la transmission du téléphone par voie numérique aboutit à deux résultats : la mise à la disposition des abonnés de lignes parfaitement adaptées aux transmissions informatiques (donc sans recours aux modems) avec des débits élevés (**Numéris**) et, d'autre part, à la mutation progressive des sociétés de téléphone en organismes de *télécommunication* capables de véhiculer, d'un bout à l'autre du pays ou même de la planète, toutes sortes d'informations, qu'elles soient d'origine vocales, informatiques ou visuelles comme la télévision.

Cependant actuellement, on rencontre encore les deux types de modulation sur ces voies de communication : la transmission directe des signaux binaires est appelée transmission en *bande de base*, le passage par la modulation d'une onde porteuse s'appelle transmission en *bande large* ! On voit que le jargon des spécialistes n'est pas des plus clairs.

La transmission en **bande de base** rencontre une autre difficulté. Les lignes longues sont toujours connectées aux *terminaux*, modems ou répéteurs par des **transformateurs** et non par une liaison directe, qui présenterait de nombreux inconvénients (conduction de l'électricité statique et des courants dus aux différences du potentiel tellurique...). Or les transformateurs ne transmettent pas la *composante continue* d'un signal. Le signal informatique doit donc, pour ne pas être déformé, ne pas contenir de composante continue (fréquence zéro). On peut dire qu'il ne doit transporter

aucune quantité de courant, c.-à-d. être "aussi souvent" négatif que positif, ou bien que sa valeur moyenne doit être nulle. Pour atteindre cet objectif, le potentiel zéro ne peut pas servir à représenter un bit : si le bit 1 est codé par la tension *V*, le bit zéro sera représenté par la tension *−V* (ou vice versa, auquel cas le principe sera qualifié de *logique négative*). Une telle convention est appelée **codage bipolaire** ou *code NRZ* (non retour à zéro). Très souvent utilisée, elle permet de se rapprocher des conditions mentionnées. Mais elle n'est pas suffisante, car elle nécessiterait l'égalité entre le nombre des bits 1 et celui des bits 0. Des codes plus complexes, comme le code *biphase* ou celui de *Miller*, donnent de meilleurs résultats.

3 - LES LIAISONS PARALLELES

Contrairement au téléphone qui n'emploie qu'un ou deux circuits *(paires)*, les systèmes de communication parallèles en utilisent un nombre *N* bien plus grand, ce qui multiplie par *N* le débit d'information. On n'emploie cette méthode que sur de courtes distances, car des difficultés surgissent rapidement.

Elles consistent d'abord dans le prix de la ligne, en gros multiplié par *N* et dans la difficulté à la blinder. En outre, des différences de vitesse de transmission entre voies entraînent des désynchronisations sur les signaux. Mais la principale raison du faible développement des communications parallèles est leur incompatibilité avec le réseau téléphonique classique.

Les liaisons parallèles se subdivisent en deux catégories : les liaisons **point à point** qui ne relient que deux appareils entre eux et les **bus**, qui en relient plusieurs. Un exemple de liaison parallèle point à point est donné par la ligne *Centronix* connectant la plupart des imprimantes aux calculateurs. Ceux-ci peuvent gérer plusieurs lignes de ce type (par des sorties appelées LPT*n*, *n* = 1,2,3... dans les IBM-PC), qui peuvent d'ailleurs *attaquer* d'autres appareils que des imprimantes, mais à raison d'un seul par ligne et à faible distance.

Le **bus parallèle** le plus connu est le *bus externe* des calculateurs (cf. chap. IV, page 28). Bien qu'il fasse partie du boîtier de la machine, on l'appelle ainsi pour le différencier du *bus interne*, situé dans la puce elle-même. Ce bus non seulement sert de lien entre l'unité centrale et les périphériques, clavier, écran, unités de disques, mais c'est sur lui que viendront se connecter les **cartes** d'entrées-sorties *(interfaces)* nécessaires à toute la télématique. Même vu seulement sous cet angle, ce bus joue un rôle primordial, spécialement dans la famille IBM-PC, dont il a contribué à assurer la suprématie.

Non seulement ce bus recevra les cartes permettant d'attaquer réseaux et lignes de communication, mais pourront s'y brancher aussi des cartes capables de traiter des informations externes diverses du plus haut intérêt. Ces cartes accepteront d'échanger des signaux avec des **appareils extérieurs**, des périphériques certes, mais aussi avec n'importe quelle machine électrique. Les appareils d'un laboratoire, ceux d'un atelier, peuvent par ce moyen être reliés à ce bus. On parle alors de *traitement en temps réel*.

Pour ce faire, de nombreuses cartes, appelées parfois *cartes d'extension*, existent dans le commerce. Elles permettent :

– de **recevoir des signaux** extérieurs **binaires** (tout ou rien), comme ceux indiquant la fermeture d'un contact, celle d'une vanne, le franchissement d'un seuil, etc,

– d'**émettre des signaux binaires** pour provoquer une action "tout-ou-rien", comme enclencher un relais, fermer un circuit, une porte ...

– de **compter** des événements ou des impulsions électriques, ou, ce qui revient au même, de mesurer leur fréquence,

– d'**envoyer des signaux horaires**, c'est-à-dire des impulsions à intervalles fixes,

– d'**envoyer des tensions analogiques** à un appareil exigeant ce type de commande (on parle parfois alors de *commande proportionnelle*). Une tension analogique est une tension dont toute valeur, comprise entre deux bornes, 0 et V ou −V et V, est significative. Ces tensions sont produites par la *conversion* d'une variable numérique manipulée par le calculateur. Exemples de telles commandes : position angulaire d'une gouverne, vitesse d'un moteur, valeur de consigne d'un asservissement.

– de **recevoir des tensions analogiques**, tensions résultant par exemple de la mesure de températures, distances, pressions... Ces tensions seront converties dans la carte en valeurs numériques par un *codeur* ou *convertisseur analogique-numérique* (ADC, *analog to digital converter*) sur *n* bits (par ex. avec *n* = 10 bits, sur 1024 niveaux, donc au millième près). Si on doit traiter ainsi plusieurs signaux de ce type variant lentement, les cartes peuvent les *multiplexer*, c.-à-d. les scruter tour à tour et en prélever à chaque fois un échantillon *(sample)* qui sera ensuite codé. Ainsi un seul codeur convertissant un échantillon en 100 µs peut multiplexer la sortie de 16 échantillonneurs mesurant toutes les 2 ms la tension de 16 *voies* analogiques **(6)**.

Les calculateurs évoluant assez vite (beaucoup plus vite en tout cas que l'installation pilotée), il s'est

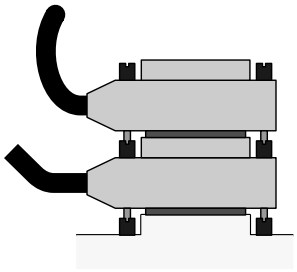
(6) La mesure étant brève, il faudra auparavant filtrer ces signaux et réduire leur bande passante à moins de 1 khz si l'on veut que la valeur mesurée ait un sens.

posé le problème de la **compatibilité** entre ces cartes et l'*ordinateur hôte*. Les cartes sont en effet prévues pour un *bus* donné. Dans le monde IBM-PC, à ses débuts, le bus (externe) était de type 8 bits et le connecteur des cartes occupait environ 10 cm (cartes courtes). Quand, pour le **PC-AT**, le bus est passé à 16 bits en doublant de largeur, on a simplement allongé le connecteur des cartes à environ 20 cm (cartes longues), les cartes 8 bits pouvant encore être connectées sur le bus 16 bits. Ce bus s'appelle **ISA** *(industrial standard architecture)*. Il fonctionne à 8 Mhz et peut débiter au maximum 8 Moctets/s. Actuellement, on passe à 32 bits. IBM a mis au point pour cela un dispositif de bus breveté (**MCA**), incompatible avec les cartes de 16 et 8 bits. Devant cet état de fait, les concurrents se sont groupés pour s'entendre sur un autre bus de 32 bits (bus **EISA**, *extended ISA)*, incompatible avec MCA, mais acceptant les cartes ISA de 8 et 16 bits, ce qui lui procurait un avantage certain sur le bus MCA. Ces deux bus, fonctionnant à 33 Mhz, peuvent débiter 132 Mø/s.

Cette *guerre des bus* n'a profité ni à l'un ni à l'autre. La nécessité de disposer d'un bus plus rapide que l'ISA au moins pour le graphique a provoqué l'essor des **bus locaux** (cf. chap. IV, §5, p. 29), qui, non normalisés au début, sont maintenant presque tous de type VESA ou PCI. Pour l'instant, les bus locaux n'autorisent pas encore le branchement de cartes d'extension. La plupart des applications industrielles se contentent de la rapidité modeste du bus ISA.

Le troisième exemple de liaison parallèle sera encore un bus. Il s'agit du bus ou *réseau* appelé **HPIB** *(Hewlett-Packard instrumentation bus)*, puis **GPIB** *(general purpose ...)*, puis IEEE488, au fur et à mesure du ralliement à son égard d'organismes de plus en plus nombreux.

Le câble GPIB se branche sur chacun avec des fiches-gigogne permettant le *chaînage* (figure ci-après). La longueur d'une liaison ne doit pas dépasser 3 mètres et la chaîne totale 20 mètres, restreignant son emploi au laboratoire ou à l'atelier. On n'est pas obligé de mettre sous tension tous les appareils reliés au bus, mais au moins la moitié environ.



C'est un bus puissant, mais délicat ; les problèmes de synchronisation et d'impédances de charge y sont cruciaux. Il peut réunir dans sa configuration la plus simple jusqu'à quinze appareils, auxquels on donne une adresse grâce à un jeu de commutateurs.

Avec ses 16 lignes d'information (protégées par 8 fils de masse, soit un total de 24 conducteurs), ce bus peut transmettre jusqu'à un mégaoctet par seconde, à raison d'un octet à la fois sur les 8 voies de données. Temporairement, ces données peuvent être des adresses. Trois lignes servent à la concertation *(prêt à recevoir, j'envoie, bien reçu)* et cinq autres au pilotage général. Ce réseau est facile à programmer, pourvu que chacun des appareils connectés, y compris le **calculateur pilote**, comporte un module *(interface)* ou une carte obéissant aux normes IEEE488.

Ce pilote, ou *controller*, peut déléguer à l'un des appareils le droit d'émettre (il devient alors *talker*). Il doit désigner ceux à qui est destiné le message *(listeners)* ; leur nombre doit en effet être restreint au minimum indispensable, car l'émetteur attend, avant la sortie de tout nouvel octet, un accusé de réception *(acquittement)* de la part de tous les destinataires de l'octet précédent. C'est pour cela qu'on parle de communication **asynchrone**, sa cadence n'étant pas dictée par une horloge, mais par la concertation.

Le bus GPIB est capable d'interruptions : un appareil peut à tout moment faire savoir au pilote par la ligne spéciale *SRQ* qu'il a un message à transmettre. Dès que le bus sera libre, le pilote demandera tour à tour à chaque appareil qui a généré l'interruption *(polling)* et l'autorisera à émettre son message. Hélas, la plus grande partie des cartes actuelles (1994) satisfait mal sur ce point la norme GPIB, ne convertissant pas en *interruption IRQ* pour le calculateur celles du bus GPIB.

4 - LA LIAISON SERIE RS232

Cette liaison est très importante à plusieurs points de vue. Elle est d'abord très générale en informatique, même les calculateurs personnels en étant munis en version de base. Ensuite, les procédés qu'elle met en œuvre préfigurent ceux utilisés dans les réseaux de grande envergure.

La principale limitation de cette liaison est d'être *point à point*, les normes auxquelles elle obéit (RS232, V24) ne prévoyant pas la transmission d'adresses. Cependant, un calculateur peut desservir plusieurs lignes de ce type, deux à quatre par exemple avec le DOS, selon sa version. Il existe de plus des cartes pour les PC qui gèrent 4 ou 8 lignes par programmation directe (en langage machine ou en C). Rien n'empêcherait un PC de desservir ainsi une centaine de liaisons RS232 et de constituer le *nœud* d'un réseau *en étoile*.

La liaison RS232 **la plus simple** exige peu de conducteurs, trois au minimum : un pour l'émission, un pour la réception et un retour commun, ou bien deux fibres optiques. Si l'on utilise les lignes téléphoniques classiques, il faudra passer par un *modem*. Le câble entre calculateur et modem comportera beaucoup d'autres fils pour assurer l'ouverture de la liaison et le pilotage du modem (c'est pour cela que connecteurs et câbles RS232 **normalisés** comportent un si grand nombre de lignes : 25 ou 9). Dans ce cas, matériel et logiciel sont fournis par la même firme et l'utilisateur débutant n'a pas à se préoccuper du mode de fonctionnement (en particulier le modem se charge de composer le numéro d'appel et d'attendre les tonalités). Une **liaison sans modem** est dite *à modem nul*. Elle est **très employée** pour raccorder de nombreux périphériques à un calculateur.

La *carte* de communication comporte une *puce* (**ERAU**, *émetteur-récepteur asynchrone universel*, UART) qui régit la transmission et *sérialise* (7) tout d'abord les octets que le bus du calculateur lui adresse. Les informations sont donc envoyées bit après bit, sous un **débit** fixé par le programme à une valeur compatible avec matériel, ligne, carte et périphérique. Les lignes téléphoniques ordinaires ne permettent que des débits moyens (de 110 à 28 800 bps, parfois plus, s'il y a compression), les commandes DOS autorisent jusqu'à 9600 bauds et la programmation directe 115 200.

Les initialisations nécessaires (débit, taille du caractère,...) sont assurées soit par des interrupteurs, côté périphérique, soit par programmation, côté calculateur. Les caractères transmis comprennent généralement 7 bits, mais peuvent être étendus à 8 si les deux correspondants sont réglés sur cette taille. La transmission étant *asynchrone* (i.e. sans synchronisation sur une horloge commune), il faut s'entendre sur le moyen de reconnaître début et fin d'un caractère. En fait, on précise l'intervalle séparant deux caractères par la notion de *nombre de bits d'arrêt*. Enfin, on peut demander une détection élémentaire d'erreur en transmission par la vérification de la *parité* de chaque octet (on détecte ainsi l'absence d'*un* bit, ou celle d'un *nombre impair* de bits).

Ce contrôle ne suffit pas. Les périphériques étant généralement lents, il faut s'assurer que les octets sont malgré tout bien "assimilés". Le récepteur dispose d'une mémoire tampon de capacité limitée et donc saturable. Il doit alerter l'émetteur de l'approche de la saturation afin que l'émission cesse jusqu'au retour à la normale. Pour ce faire, plusieurs méthodes de concertation peuvent être employées. Elles sont étudiées dans l'annexe 2 de manière assez générale, le périphérique ne modifiant guère le problème posé. Les deux principales méthodes sont :

– la **régulation câblée**, qui nécessite une ligne supplémentaire. C'est en maintenant celle-ci à 0 ou à 1 que le récepteur demande l'arrêt ou la reprise de l'émission.

– la **régulation *Xon-Xoff*** qui consiste, pour le récepteur, à envoyer sur sa ligne d'émission un caractère spécial (ASCII n° 17 ou 19, appelés *Xon, Xoff)* pour en demander l'arrêt ou la reprise. Cette méthode est moins rapide et plus délicate que la première (risque de pertes des caractères de commande), mais n'exige pas de ligne supplémentaire.

Le circuit émetteur de l'ERAU n'est pas agencé (*câblé*) pour prendre en compte toutes les précautions nécessaires. Il appartient au programmeur – au concepteur du logiciel de communication – de vérifier avant d'envoyer un caractère que le récepteur peut le recevoir. Il devra également vérifier que l'émetteur lui-même est libre, car le temps mis pour *sérialiser* et expédier un octet n'est pas négligeable.

En réception, l'ERAU se comporte de manière désagréable : si l'on cherche à lire dans son registre d'arrivée un octet non reçu, le calculateur *se plante*, c'est-à-dire attend indéfiniment l'arrivée de ce caractère. Or il existe un bit dans le registre d'état de l'ERAU qui témoigne si un octet a été reçu. Le programmeur doit aller lire l'état de ce bit avant toute tentative de lecture de caractère, y compris celle des Xon-Xoff éventuels. Cette dernière lecture remet à zéro le registre de réception. Encore une précision : si un caractère est reçu avant la lecture du précédent, celui-ci est perdu (*écrasé*), ce que signale encore le mot d'état.

La liaison RS232 fait preuve de toute son efficacité lorsque sa partie réception est autorisée à provoquer une interruption dans le calculateur. Détailler cet aspect nous entraînerait trop loin; une programmation en langage évolué avec les précautions énoncées plus haut permet déjà d'obtenir sur cette ligne des communications rapides et sûres.

(7) Sérialiser un octet veut dire transformer l'ensemble des 8 bits arrivant *simultanément* par le bus en une suite (une *série*) de 8 bits sur un seul fil.

5 - LES RESEAUX LOCAUX

Si l'on veut relier un grand nombre de postes informatiques, il faudra songer à construire un réseau local *(LAN)*. Pour la mise en commun d'un périphérique (imprimante par exemple), on peut se contenter de solutions peu onéreuses disponibles dans le commerce. Mais un partage plus vaste de ressources diverses (périphériques spéciaux, fichiers importants, bases de données...) exigera un vrai réseau.

Un réseau local est un réseau **limité géographiquement** au domaine privé, celui d'une entreprise par exemple. Il comportera des *lignes*, telles celles décrites § 1, mais de bande passante élevée, torons de paires blindées, câbles coaxiaux **(8)**, ou fibres optiques. Sa configuration pourra être en *anneau (ring)*, en *étoile (star)*, en *arbre*, en *chaîne* (*daisy chain, guirlande)*, en *bus* ou *maillée* (fig. 14-4). Aux points de jonction des lignes, il comportera des **nœuds**, qui sont parfois de véritables ordinateurs (appelés *hubs* dans certains réseaux). Les appareils reliés au réseau s'appellent **stations**, **terminaux** ou **postes**. Au point de vue technique, l'acheminement des signaux se fait toujours en série, le plus souvent en bande de base, avec un débit dépassant un mégabaud et sous un code NRZ ou plus élaboré (code de *Miller* ou *bipolaire*).

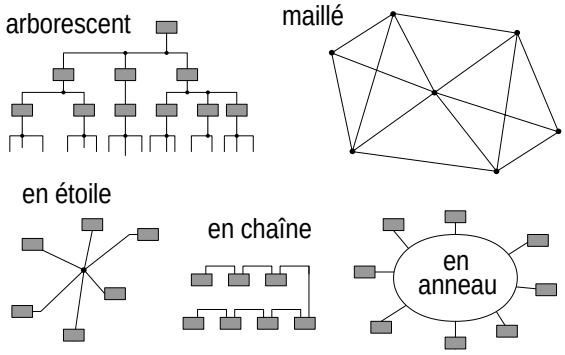


Fig. 14-4 : *Topologie de réseaux. La configuration en bus manque dans l'inventaire ; son dessin serait analogue à celui de l'anneau, mais ne serait pas fermé.*

Dans les réseaux, un message comprend toujours l'adresse du destinataire. Il est acheminé de nœud en nœud et mémorisé à chacun d'entre eux. Dans un réseau maillé, il peut suivre des chemins imprévisibles. Ce sont les nœuds qui choisissent sa *route* en fonction de l'état actuel du réseau (encombrement des lignes, des nœuds, défaillances, ...). Le destinataire rend toujours compte à l'expéditeur de la réception correcte du message. Des procédés plus complexes que l'examen de la parité permettent de détecter si le message reçu est bien celui émis. Certains de ces procédés sont capables de régénérer un ou plusieurs bits erronés **(9)**.

Pour accroître l'efficacité du réseau, on découpe les messages en **paquets** ou *trames* (d'un kilo-octet par ex.), précédés d'informations, adresse du destinataire, numéro du paquet... Ce numéro sert à reconstituer l'ensemble du message, puisque les paquets qui le composent, empruntant des routes différentes, ne parviennent pas toujours dans le bon ordre au destinataire (dans un réseau maillé seulement).

Le **logiciel** gérant une telle communication est donc complexe, d'autant plus que les postes reliés peuvent être de natures diverses (réseaux *hétérogènes)*. Des réseaux de types différents peuvent même être reliés par des nœuds spéciaux appelés *passerelles*. De tels résultats n'ont pu être obtenus qu'en astreignant les logiciels à obéir à des procédures de communication sévères **(protocoles)**. Pour s'adapter à la diversité des moyens, ils sont divisés en *couches logiques* (au nombre de sept dans la norme OSI) qui, tour à tour, mettent en forme le message jusqu'à son envoi dans la ligne.

Pour l'utilisateur, le réseau se traduit par un branchement via un câble sur l'un de ses *accès*, l'installation d'une *carte* spécifique dans son ordinateur et l'implantation d'un logiciel de gestion. Nous ne dirons que quelques mots des deux types de réseaux locaux les plus répandus, les réseaux à *accès aléatoire* et les *anneaux à jeton*.

Les **réseaux d'accès aléatoire** ont pour ancêtre (1971!) le réseau radio ALOHA reliant les centres universitaires des îles Hawaï. A l'origine, chaque station envoyait un message quand bon lui semblait. Si la fréquence (le *canal)* était déjà occupée, il y avait *collision de messages*. On les détruisait et on recommençait après un délai **aléatoire**, variable heureusement d'une station à l'autre. C'est le protocole *ALOHA pur*. L'efficacité était faible (au mieux, le canal ne peut transmettre de messages valides que pendant $1/2e \approx 18\%$ du temps).

On améliore le procédé par la synchronisation du début des émissions *(slotted* ALOHA, gain de 2 en efficacité), ou par l'**écoute du canal** (**CSMA**, *carrier sense multiple access)*. La station voulant émettre commence poliment par écouter la ligne et n'émet que lorsqu'elle n'entend plus rien. Hélas, deux stations peuvent en faire autant au même moment : il y a encore collision ; on détruit le message à la fin de l'envoi et chacune reprend comme ci-dessus. Avec le protocole **CSMA-CD** (*collision detection)*, chaque émetteur s'arrête *dès qu*'il a détecté une collision.

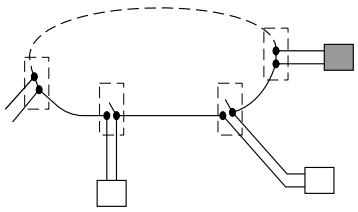
(8) Le diamètre, la qualité des câbles et celle des connecteurs, jouant sur l'atténuation, sont fonction de l'étendue du réseau. Le prix de revient s'en ressent.

(9) La détection de *parité* consiste à émettre un bit excédentaire permettant au destinataire de savoir si le message initial comportait un nombre pair ou impair de bits. On conçoit que *n* bits supplémentaires permettent la déduction de *n* informations, *redondantes* en général, mais autorisant des corrections d'erreurs.

Ethernet (de Xerox) est le plus connu des réseaux de ce type. Ses caractéristiques (étendue de 2 à 3 kilomètres, débit 20 Mbd, connectable à plus de 1000 postes) en font un réseau local haut de gamme. D'autres types de réseau ont des atouts plus modestes et sont mieux adaptés si le nombre de stations est réduit (100 par exemple). Leur topologie est de type *bus*, reliant des étoiles, parfois avec maillage.

De préférence, Ethernet utilise un câble coaxial de haute qualité, mais du coaxial normal (avec prises BNC) suffit pour de courtes liaisons *(Ethernet fin)*. Un émetteur-récepteur HF *(transceiver)* peut être fixé sur le câble par perforation ; il sera relié par 5 paires aux stations qu'il dessert. Parmi les avantages de ce réseau, on note son aptitude à l'insertion de nouvelles stations sans interruption du service et sa faible vulnérabilité aux coupures de câble. Parmi ses défauts, on cite d'abord le caractère aléatoire de son temps d'accès (en théorie, une station peut attendre indéfiniment la possibilité d'émettre, ce qui est inacceptable dans certaines applications industrielles). On cite également sa faible efficacité (ce qui oblige à le surdimensionner), son risque d'engorgement et la difficulté qu'on y trouve à localiser les coupures de câble.

L'**anneau à jeton** est un réseau série dont les stations sont connectées sur les *nœuds* d'une seule et même artère, par des lignes limitées en longueur appelées *lobes* (par ex. : le *Token Ring* d'IBM). Les nœuds peuvent être physiquement groupés (souvent par 8) dans un coffret. Ces coffrets sont *chaînés* de façon à réaliser un anneau fermé. Le rôle de ces nœuds est modeste : ils servent surtout à ouvrir l'anneau pour y inclure la station raccordée quand elle se met en service et à le refermer en court-circuitant son lobe quand elle se retire. Ce sont les *cartes* de la station qui assurent le rôle de sélection et de *répétition* dévolu aux nœuds dans les autres réseaux. Ces cartes commandent les relais qui ouvrent le lobe dont elles dépendent (figure ci-dessus). En fait, une station en train d'émettre *ouvre temporairement* l'anneau, étant la seule dont l'émetteur ne *répète* pas ce que reçoit son récepteur.



Ce réseau est caractérisé par la présence d'un *jeton*, paquet logiciel véhiculant de poste à poste le *droit à émettre*. Seul le poste détenant le jeton peut émettre. Le risque de collision serait nul si on n'observait pas parfois la présence de plusieurs jetons. Cette anomalie est détectée par le **surveillant** du réseau, qui détruit les jetons excédentaires, ou bien en crée un si le premier disparaît. Ce rôle de *cerbère* est dévolu à l'un des postes en service. S'il vient à être mis hors service, une procédure permet *l'élection* d'un nouveau surveillant. Aux faibles débits, l'anneau à jeton est moins efficace que celui à accès aléatoire, puisqu'en moyenne avant d'émettre, un poste doit attendre un délai voisin de la moitié du temps de parcours de l'anneau. Mais ce temps d'attente est borné, ce qui est un avantage, comme l'est le fait qu'aux débits élevés, ce réseau n'est pas handicapé par les collisions et ne court pas le risque d'*engorgement* inhérent à ceux du premier type. Il est par contre très sensible aux coupures (d'où difficulté d'insertion sans interruption du réseau), mais ces coupures sont faciles à détecter.

Le désir d'homogénéiser matériel et logiciel employés sur ces réseaux a entraîné la publication de normes. Les normes IS 8002 . *x* ou **IEEE 802 .** *x* s'appliquent aux réseaux locaux ($x=3$ pour ceux de type CSMA-CD, $x=5$ pour les *anneaux à jeton* et $x=4$ pour ceux de type *bus à jeton*, non décrits ici).

Le même désir, plus le besoin d'interconnexion, a suscité la normalisation du logiciel d'accès à ces réseaux. On a mentionné la norme **OSI** *(open system interconnection)* de l'ISO et sa décomposition des fonctions en 7 couches logicielles. Ce modèle est cependant peu suivi, le protocole voisin, **TCP-IP** *(transmission control process - Internet protocol)*, étant beaucoup plus souvent utilisé, car il assure découpage en paquets, transmission et réassemblage avec beaucoup de fiabilité.

6 - LES GRANDS RESEAUX

Par grands réseaux *(WAN)*, on entend ceux débordant le domaine privé. Ils sont parfois publics, mais le plus souvent leur *usage* demeure restreint aux organismes qui les ont créés (industriels, banques, universités, administrations...). Malgré tout, dans tous les cas (hormis peut-être celui des Armées), leurs supports ne leur sont pas propres. Ce sont ceux gérés par les établissements de télécommunications, tels que France-Télécom ou les PTT. Ces supports, allant des paires téléphoniques aux satellites, ont été inventoriés § 1.

Ces réseaux sont toujours *maillés* ; les problèmes de choix de la *route* et de temps de propagation s'y posent avec acuité. On sait qu'un destinataire envoie toujours à l'émetteur un compte rendu de réception. Celui-ci ne parvient qu'au bout de délais prohibitifs si le message s'en va à l'autre bout de la Terre ou s'il passe par un *satellite* (dans ce cas, délai d'au moins 0,27 s). Aussi, des procédés appropriés doivent-ils être employés comme celui de la **répartition temporelle** des accès, chaque demandeur se voyant attribuer périodiquement une tranche de temps courte pour son émission. On parle de *multiplexage temporel*.

Différents réseaux sont proposés aux informaticiens par les services publics de télécommunications. On essayera de les classer en trois catégories : les réseaux *alloués*, les réseaux *commutés* et ceux à *transmission par paquets*.

Des artères à grand débit peuvent être *louées* à un demandeur *(spécialisées* dans la terminologie des PTT), pour relier en permanence, durant une longue période, deux établissements ou plus. C'est une solution forcément onéreuse, car elle "privatise" en quelque sorte un bon nombre de *voies* téléphoniques. Elle est parfois indispensable pour relier les réseaux locaux de deux sites éloignés d'une même entreprise.

A l'autre extrémité de l'inventaire, on trouve le réseau téléphonique normal, dit **commuté** ou RTC, accessible aux transmissions informatiques par des modems (étudiés §2), mais bon marché si le volume des données transmises est faible. Comme chacun sait, toute demande d'accès à ce réseau se formule en décrochant le *combiné* et en composant un numéro (le modem le fait pour vous). Dans les centraux téléphoniques *(autocommutateurs)*, des circuits logiques – actuellement des *micro-processeurs* – sélectionnent la meilleure route à suivre et établissent un lien physique temporaire entre les deux correspondants en commutant des tronçons les uns à la suite des autres. Cette *route* est allouée au demandeur jusqu'à la fin de sa communication. Entre villes, la route empruntera l'un des canaux d'une artère de haut débit (câble, faisceau...) : même dans ce mode, il y a partage des moyens.

Mais il existe également des réseaux purement télématiques, constitués de tronçons reliés en permanence et maillés entre eux. Les messages qu'ils acheminent sont toujours décomposés en **paquets**. Chacun d'eux suit une route qui lui est attribuée par les nœuds du réseau. Ces routes sont complètement banalisées, les paquets successifs n'ayant ni même origine, ni même destination, de sorte que les temps de silence éventuels d'un dialogue sont alloués à un autre demandeur. La rentabilité des lignes y gagne considérablement. Dès que les volumes de données à transmettre sont importants, on a intérêt à choisir ce genre de réseaux, pour lesquels on payera un droit de raccordement, un abonnement et des redevances tenant compte du temps de connexion, du volume transmis, mais peu de la distance.

Sur certains de ces réseaux, des *serveurs* sont raccordés et accessibles à quiconque acquitte les droits de consultation. Ces serveurs gèrent des *bases de données* contenant des informations utiles à certaines professions ou même au public. En France, une très grande partie de ces serveurs est connectée au réseau **Transpac**, réseau public maillé de faible débit (4800 bauds) travaillant par paquets, comme son nom l'indique. Le Minitel est également un réseau, dont le très faible débit suffit à certaines applications (messagerie), mais qui est connecté à Transpac pour sa mise en relation avec ses serveurs les plus puissants.

Bien qu'on ait nettement distingué réseau commuté et réseau à paquets, l'évolution en cours tend à rapprocher ces notions, puisque les grandes artères du réseau commuté sont en majeure partie numérisées et banalisées. Si autrefois les différentes conversations téléphoniques étaient transmises chacune sur une ligne donnée ou sur une bande de fréquence donnée (partage de bande), on évolue vers un emploi de ces lignes en *bande de base* avec **partage du temps** : chaque *échantillon* d'une conversation entre dans une artère et occupe un faible créneau temporel. Les paquets sont suffisamment petits pour que l'intervalle entre ceux d'une même conversation ne soit pas sensible aux auditeurs. La parole est pour cela découpée à raison de 6 à 8000 échantillons par seconde **(10)** dont l'amplitude sera codée sur 8 bits qui, sérialisés, donneront lieu à un débit de 32 à 64 kb/s. Faciles à réamplifier avec un excellent rapport signal sur bruit (comme toute information numérique), ces transmissions s'avèrent être de qualité remarquable, même à grande distance. Ce sont elles qui assurent actuellement les communications lointaines.

Cette évolution amène les services de télécommunication à offrir à leurs clients des connexions entièrement numériques. C'est le réseau **RNIS** (numérique à intégration de services) ou **Numéris**, auquel peuvent se raccorder sans modem aussi bien ordinateurs et télécopieurs (numériques eux aussi) que les téléphones de la nouvelle génération. En France, la ligne élémentaire s'appelle *accès de base* et comprend deux canaux de 64 kilobauds pour les données ou la voix et un de 16 pour la messagerie (Minitel) et les services supplémentaires (double appel, facturation, identification de l'appelant...). Cette évolution consacre la mainmise totale de l'informatique sur les systèmes de communications.

(10) D'après le 2e théorème de Shannon, on peut reconstituer tout signal sinusoïdal s'il a été échantillonné à une cadence au moins double de sa fréquence propre, donc tout signal échantillonné au double de sa fréquence maximum : 3400 hz pour le téléphone.

7 - INTERNET (11)

L'un des premiers grands réseaux expérimentaux a probablement été *Arpanet*, qui reliait dès 1969 des centres de recherches avancées travaillant pour l'armée aux USA. Son architecture était maillée et auto-reconfigurable : en cas de conflit, même avec beaucoup de lignes détruites, les communications empruntent automatiquement un autre chemin (on donne souvent deux exemples récents montrant la *robustesse* de ce type de réseau : lors du séisme de Los Angeles en 1994, ou du coup d'état de Moscou, Internet a été le seul moyen de communiquer avec ces sites alors coupés du monde).

Dans les années 80, les universitaires avaient commencé à relier les réseaux locaux des campus par de grands réseaux. Le plus connu est Bitnet, qui s'était même étendu à l'Europe (sous le nom d'EARN), promu et financé par IBM (il est maintenant en déclin sous la poussée d'Internet). En 1990, les militaires (pour se consacrer à leur réseau MILNET) ont laissé aux universitaires Arpanet, trop expérimental, avec le protocole de communication qui faisait sa force, TCP-IP.

Arpanet, appelé depuis Internet, a eu alors une croissance spectaculaire. Il relie maintenant dans le monde entier des réseaux locaux, régionaux ou nationaux, travaillant avec le même protocole TCP-IP. C'est un fédérateur de réseaux, qu'on appelle souvent *le Net*, ou *CyberSpace*, *Cybermonde*... et qui re groupe fin 1994 environ 30 millions d'utilisateurs et 40 000 serveurs.

Le cœur d'Internet est formé de nœuds reliés par des lignes permanentes (spécialisées) à moyen ou haut débit. Le maillage de ces parties centrales est de règle. Les nœuds sont des *routeurs*, calculateurs spécialisés dont le rôle principal est l'aiguillage *(routage)* des paquets d'information entre les différentes liaisons.

Les réseaux locaux des entreprises, des campus,... peuvent se relier à cette infrastructure par liaison spécialisée. Les ordinateurs isolés (au travail ou à domicile) peuvent s'y connecter via des points de concentration (**points d'accès**) et en général le réseau téléphonique. Tous les utilisateurs raccordés (30 millions!) peuvent utiliser les services décrits par la suite.

Sur les réseaux locaux connectés à Internet, certains ordinateurs ont de grosses puissances de calcul ou hébergent de grandes bases de données, ressources pouvant intéresser des tiers. Leurs possesseurs peuvent décider de les rendre accessibles à tous, via Internet, gratuitement ou avec facturation. Ces ordinateurs sont appelés **serveurs** (de calcul, de données, serveurs FTP *anonymous*, Gopher...)

Une bonne partie des réseaux locaux des universités et centres de recherches sont réunis par de grands réseaux dans la plupart des pays. En France, le plus important d'entre eux est RENATER (réseau national pour la technique, l'éducation et la recherche) ; il fédère lui-même des réseaux régionaux comme RERIF pour l'Ile-de-France, ARAMIS pour Rhône-Alpes,... ou des réseaux d'organisme (CEA par exemple). Les réseaux régionaux interconnectent campus, écoles et centres d'études des principales villes d'une région.

L'ensemble mondial de ces nœuds, ces liaisons, mais aussi ces réseaux locaux interconnectés, forment Internet, véritable toile d'araignée sur les fils de laquelle transitent des informations par paquets sans autre intervention humaine que celle de l'utilisateur émetteur.

L'évolution technique d'Internet est assurée de manière collégiale, sous la directive d'une société savante très ouverte, l'ISOC, avec la participation de spécialistes de haut niveau, bénévoles ou temporairement mis à la disposition d'Internet par un organisme public.

Internet, d'origine universitaire, est ouvert aux industriels, aux entrepreneurs et à tout un chacun sur n'importe quelle machine. Ses fondateurs espèrent que les nouveau-venus respecteront l'éthique qui a prévalu jusque là : ni censure, ni outrance *(publicité* au sens actuel (12) du terme par exemple).

Il n'y a ni gestion centralisée d'Internet, ni droit d'adhésion unique. Chacun paie une part de la toile d'araignée. Ainsi RENATER est financé par les organismes d'enseignement supérieur et de recherche, les réseaux régionaux par les universités ou les laboratoires avec l'aide des collectivités locales.

L'utilisateur isolé, sans réseau local, passera donc par un *prestataire* de services *(provider)*. Après passation d'un contrat, il se reliera à une *machine* de celui-ci (**point d'accès**) par modem et liaison commutée (téléphonique, numéris) ou louée (Transfix, Transpac). Il bénéficiera de certains ou de tous les services du Net, avec une rapidité liée à sa ligne. Ces prestataires peuvent offrir des services supplémentaires, par exemple des serveurs spécialisés. Le prix demandé dépendra de ces services, du volume de l'information transmise, du temps de connexion, avec des forfaits annuels ou mensuels, ou bien des abonnements.

Certains prestataires privés peuvent aider leurs clients à installer les logiciels nécessaires sur leur

(11) Les sections 7 et 8 (sur Internet), ont été rédigées avec l'aide de Jean-Luc Archimbaud, ingénieur à l'Unité Réseaux du CNRS (UREC). Pour plus de détails sur Internet, on pourra se reporter à ses articles dans l'ouvrage *L'Internet professionnel*, Editions du CNRS.

(12) Par contre, la publicité au sens étymologique (se faire connaître, faire connaître ses travaux, ses produits...) est tout à fait admise.

ordinateur, à trouver les serveurs et les bases de données qui les intéressent et aussi leur offrir certaines protections : autant de *valeur ajoutée* qui se paie.

Si Internet est doté d'une grande fiabilité sur la qualité de ses liaisons, il n'est pas considéré comme très sûr en ce qui concerne ses utilisateurs et les informations qu'on y trouve. Tout le monde ayant le droit d'y écrire à peu près n'importe quoi, il faut savoir en relativiser les données. De même, le Net peut constituer un tremplin pour des fraudeurs cherchant à s'immiscer sur des machines *sensibles*. Il existe des solutions techniques en matière de protection, mais, difficiles à mettre en œuvre, elles ne sont pas monnaie courante. La sécurité est souvent citée comme étant le talon d'Achille du Net.

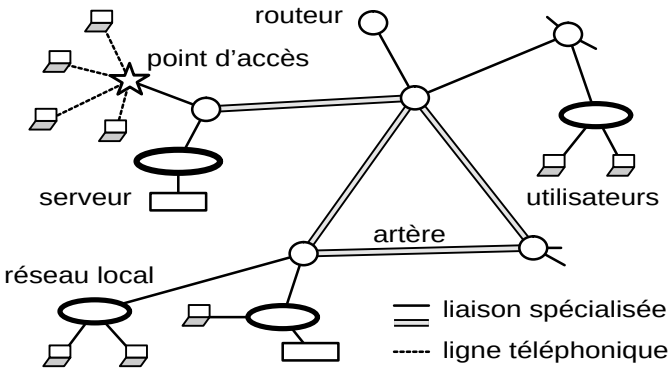


Schéma symbolique d'Internet.

8 - EMPLOI D'INTERNET

Sur Internet, chaque ordinateur (personnel, grosse machine, station de travail) reçoit un numéro d'identification (*adresse IP* comme **193.201.23.5**) et un nom unique (comme **isis.urec.fr**) qui l'identifient sans ambiguïté. Ce nom est formé de plusieurs parties, chacune séparée par un point. La dernière partie indique soit le pays (**fr**, **be, de**, **ch**, **ca** pour France, Belgique, Allemagne, Suisse, Canada), soit, aux USA, le type d'organisme : **edu**, **gov**, **mil**, **com**, **net**, **org**, **int** pour, respectivement université, gouvernement, armée, entreprises commerciales, opérateurs de réseaux, organisation d'Internet, organismes internationaux... La partie précédente (ici **urec**) indique l'entreprise, le labo, l'école... et la première (**isis**) le nom de l'ordinateur. Un utilisateur d'Internet est identifié par son **adresse électronique** formée de son nom, puis du caractère arobasse @ (*at* en anglais), puis du nom de *sa machine* (ex.: **Jean-Luc.Archimbaud@isis.urec.fr**).

Sur son ordinateur, l'utilisateur devra disposer du logiciel de communication TCP-IP et de logiciels propres à chacun des services étudiés ci-après. Ces logiciels, souvent gratuits, ont un nom qui rappelle le service auquel ils donnent accès. On peut généralement les acquérir gratuitement sur Internet via *ftp anonymous* (cf. ci-après).

Courrier électronique

Avec un logiciel comme **Eudora** (pour Mac ou PC), on peut envoyer des messages à tout utilisateur relié à Internet si on connaît son adresse électronique. Ce message parviendra en un temps très court à la *machine* dont dépend le destinataire ; il y sera mis en mémoire (c'est une boîte aux lettres) et le destinataire pourra le lire quand il se connectera sur cette machine. Certains logiciels (systèmes multitâches) sont capables d'avertir l'usager sur son ordinateur dès l'arrivée d'un courrier. C'est un mode de correspondance dont on dit qu'il est *sans contrainte de temps ni d'espace*.

L'inconvénient majeur de ce courrier anglophone est sa restriction aux caractères ASCII 7 bits. Des remèdes logiciels existent et seront bientôt utilisables par tous. En attendant, il faut ou bien ne pas écrire de lettres accentuées dans les messages, ou bien les convertir en fichiers 7 bits avec un logiciel approprié (**binhex** sur Mac, **zip** sur PC, **uuencode** sur Unix).

Forums

Avec des logiciels similaires à ceux de la messagerie, on peut envoyer des messages à des *forums (news)*, boîtes à lettres géantes dans lesquelles tout un chacun – ou bien les seuls adhérents du groupe – peuvent lire les idées des autres ou déposer leurs propres commentaires **(13)**. Dans chaque forum, les messages sont relatifs à un thème commun, qui est souvent bien loin d'être technique ou scientifique. Politique, arts, cuisine, ésotérisme... font l'objet de tels forums qui passionnent ceux qui désirent échanger leurs idées. Une coutume amusante économisant la frappe au clavier veut qu'on indique ses états d'âme par des symboles conventionnels (*smiles*) comme ;-) :-) :-Q ... pour représenter un clin d'œil, un sourire, un éclat de rire ... (en les faisant pivoter de -90°, on verra qu'ils figurent un visage).

Du fait de leur abondance, les messages dans les forums sont détruits au bout d'un délai fixé (souvent 14 jours) pour ne pas saturer les disques. Certains pensent que ces forums constituent l'apogée de la liberté d'expression : plus besoin d'imprimerie ou de *ronéo* pour diffuser ses idées ; il faut quand même un minimum d'informatique.

(13) Dans certains forums, un censeur (ou *modérateur*) opère une sélection sur les messages.

Echange de fichiers

Après le courrier, le service le plus simple est **FTP**, *file transfer protocol*, qui permet d'échanger des fichiers entre machines. Il permet surtout d'importer un fichier disponible sur un serveur *public* (i.e. libre d'accès), appelé alors **FTP anonymous**. Si on dispose du logiciel correspondant, on tapera

ftp *nom_du_serveur*

Ce nom, avec la forme décrite précédemment et qu'il faut connaître à l'avance, commence d'ailleurs souvent lui-même par **ftp**. Le serveur demandant le nom de l'interlocuteur, on répond **anonymous** (c'est une convention). A la demande du mot de passe, on donne son adresse électronique complète. On va pouvoir alors naviguer de répertoire en répertoire dans le serveur : à chaque instant, l'un de ses répertoires sera actif (cf. chap. VIII, §5) ; on va appeler *rep* ce répertoire et *lrep* (répertoire local) celui qui est actif dans l'ordinateur de l'usager. On pourra taper l'une des commandes (dérivées d'Unix) propres à **ftp**, par exemple :

pwd : affiche le nom du répertoire actif *rep* du serveur;

cd .. , **cd** *rep* : comme décrit dans le chap. VIII (§6) ;

ls ou **dir** (plus complet) : fait afficher la liste des fichiers du répertoire actif *rep* ;

more : initialise un mode de fonctionnement pour **dir** qui fait afficher écran par écran (en général indispensable) ;

ldir : permet de lire la liste des fichiers de *lrep* ;

get *fichier* : copie le contenu de *fichier* depuis *rep* dans *lrep* ;

put *fichier* : exécute l'inverse (cette commande est en général refusée par le serveur *ftp anonymous* s'il n'est pas destiné à recevoir des fichiers).

binary : initialise le mode de transmission binaire, qui seul permet de transmettre des exécutables.

Pour les opérations d'importation-exportation, il est nécessaire de disposer du droit correspondant. En général, les répertoires dont le nom contient le mot **pub** *(public)* sont ouverts à tous en lecture. Avec la commande **dir**, on lira en tête de chaque fichier, un groupe de 9 lettres donnant les *droits d'accès* au dit fichier à la mode Unix. Les 3 dernières lettres constituent les autorisations *anonymous*. Elles peuvent être : –, **r**, **w**, **x**, ce qui veut dire respectivement aucun droit, droit de lire, d'écrire, d'exécuter (pour le fichier concerné).

Dans ces répertoires, on cherchera la présence d'un fichier **INDEX**, **README**, **00README**..., qui contient la liste des fichiers avec parfois un commentaire. Il vaut mieux commencer par importer **INDEX** pour le lire à son aise. On ne peut pas lire directement sous *ftp* un contenu de fichier. Il faut l'importer, puis appeler un *éditeur* ou traitement de texte personnel.

Ces fichiers sont souvent comprimés. Ils se décompriment avec un utilitaire public adapté à leur mode de compression. La liste suivante donne, en fonction du *suffixe* du fichier, le monde dont il provient et l'utilitaire de décompression.

.zip	(PC)	**pkunzip** ou **unzip**
.hqx	(Mac)	**binhex** ou **StuffitExpander**,
.Z	(Unix)	**uncompress**,
.tar	(Unix)	**tar**.

Pour communiquer un fichier à un correspondant, on peut soit le copier sous *ftp* (avec **put** *fichier*) dans un répertoire d'une machine accessible aux deux interlocuteurs, soit l'envoyer par courrier après l'avoir encapsulé par un logiciel (conversion 8 bits – 7 bits).

Archie

Généralement, l'utilisateur ne connaît pas le serveur contenant les documents qu'il recherche. Il peut alors interroger, avec un logiciel spécifique, comme **Anarchie** sur Mac, **xarchie** sur Unix, **wsarchie** sur PC, un serveur particulier nommé **Archie**. Il donne à ce serveur le nom ou une partie du nom des documents qu'il cherche. **Archie** lui répond par la liste des serveurs *ftp anonymous* où se trouvent de tels documents (en indiquant répertoire et nom des fichiers).

Exécution à distance

Un utilisateur ne disposant pas sur sa machine d'un logiciel qui l'intéresse peut, à travers Internet, accéder à une machine distante disposant de ce logiciel et y travailler comme si son ordinateur n'était qu'une simple console *(terminal)* reliée à cette machine. Il faut que le propriétaire de la machine distante lui donne l'autorisation (en l'enregistrant et en lui donnant un mot de passe). Cette fonction est appelée *terminal virtuel* ou *distant*, ou **telnet**.

Sur son ordinateur, l'usager tapera la commande :

telnet *nom_de_la_machine*

Il frappera ensuite, à l'invite du système, nom et mot de passe, puis il enverra des commandes dans la syntaxe propre à la machine qui les exécutera.

Gopher

C'est un service d'Internet qui dispense de connaître le nom des serveurs, comme l'oblige *ftp anonymous*. Il présente tous les serveurs d'information d'Internet comme un immense livre. Il suffit d'avoir le logiciel correspondant (**TurboGopher** sur Mac, **Hgopher** sur PC, **xgopher** sur Unix) et de le configurer en lui indiquant le nom d'un *serveur-gopher* initial. Lorsqu'on lance le programme, une page, envoyée par

ce serveur initial, s'affiche sur l'écran, généralement un *sommaire*. Il suffit de cliquer sur une de ses lignes pour voir s'afficher un document ou passer à un autre sommaire et réitérer. Les pages peuvent provenir de n'importe quel serveur sur Internet. On peut ainsi, de référence en référence, parcourir l'immense livre – certains disent *naviguer* – en accédant à différents serveurs sans connaître à l'avance ni leur nom, ni leur adresse.

Au cours d'une session **gopher**, on peut revenir sur les pages vues auparavant, constituer une liste d'intérêt en marquant des éléments reçus **(14)**, revenir sur cette liste...

Veronica

C'est pour *Gopher* l'équivalent d'*Archie* pour *ftp*. L'utilisateur interroge un serveur *Veronica*, avec le même logiciel que celui utilisé pour *Gopher*. Il indique ce qu'il cherche sous la forme d'une expression logique contenant mots-clés et opérateurs booléens.

Wais

Il s'agit d'un service qui, avec des logiciels tels que **WinNais** pour Windows et **MacWais** pour Mac, permet d'accéder à des bases de données indexées, situées sur des serveurs d'Internet. C'est un service très utilisé par les documentalistes qui ont ainsi classé leurs fiches et les ont rendues accessibles par les réseaux. Lorsque l'utilisateur interroge une base Wais sur un serveur distant, il donne une suite de mots-clés. Le serveur lui répond en proposant une liste de documents (avec résumé) satisfaisant à sa demande. L'utilisateur peut alors rapatrier tout ou partie de ces documents et les étudier sur son ordinateur.

Il existe un annuaire mondial des *bases wais*, **directory-of-servers**, sur le serveur **quake.think.com** et un annuaire des bases françaises sur la machine **zenon-inria.fr**.

WWW

WWW est un protocole qui permet d'accéder à un ensemble de serveurs contenant des fichiers de type *hypertexte*. Dans ces fichiers, on trouvera des mots-clés affichés en surbrillance qui constituent des liens vers d'autres fichiers ou sommaires. Si on clique sur ces mots, on verra afficher soit un sommaire, soit une liste de fichiers, soit directement le fichier lié s'il est unique ; tout ceci ayant rapport avec le mot-clé. Ces fichiers peuvent contenir texte, images ou sons. Le service rendu va dans le même sens que celui de **gopher** (navigation et *multimédia*), mais il est beaucoup plus puissant et convivial. Pour le mettre en œuvre, il faut des moyens importants, terminal graphique X ou Windows, ligne de bon débit et bien sûr la souris. On accède à un serveur WWW avec des logiciels comme **mosaic**.

Ce sont des informaticiens du CERN qui ont élaboré WWW et leur serveur **www.info.cern.ch** est un point d'entrée possible pour ce service. Signalons aussi le serveur du ministère de la Culture **www.culture.fr** où l'on pourra obtenir par exemple des extraits des collections des musées de France.

Browser

Il existe maintenant des logiciels très puissants et très conviviaux appelés *Browser*, dont les plus connus sont **Mosaic** et **Netscape** et dont des versions gratuites existent pour PC, Mac et Unix. Ils intègrent dans un logiciel unique les fonctions d'accès aux serveurs WWW, *ftp anonymous*, Gopher, *wais* et aux *forums*. Ce sont ces logiciels qui ont rendu Internet si populaire et accessible à l'utilisateur non informaticien.

Prestataires Internet

Il s'agit soit de sociétés commerciales, comme :

RAIN (Transpac), av. du Maine, Paris
Calvacom, filiale de Compuserve (US), 78140 Velizy
Oléane, filiale d'Apysoft et Pipex, 94300 Vincennes
INBE, filiale belge de Pipex (GB)
FranceNet, r. du Fbg Poissonnière, 75009 Paris
WorldNet, 77500 Chelles

soit d'associations ou assimilées, comme :

RENATER, Université de Jussieu, Paris,
French Data Network, rue Belgrand, 75020 Paris
FNET-EuNet France, av. Gde Armée, 75017 Paris

On peut consulter aussi 3615 INTERNET.

(liste très succincte, donnée à titre indicatif)

(14) En cliquant sur le bouton approprié. Si le logiciel n'est pas graphique, taper **a** (pour ajout), **d** (supprimer), **v** (voir la liste), **u** (voir la page précédente).

9 - RECOMMANDATIONS

Dans ces cinq derniers chapitres, on a vu ce qu'était la bureautique, cette branche de l'informatique la plus accessible au profane et la plus populaire. L'auteur tient à préciser qu'il a soigneusement voulu éviter de mettre des noms aux logiciels qu'on vient d'étudier. Ils sont en effet très nombreux, surtout ceux pour les IBM-PC, et leur étude exhaustive est délicate. Parfois des revues spécialisées consacrent un article à ce genre d'exercice et on voudra bien s'y reporter. On a mis l'accent sur les caractéristiques générales et le lecteur jugera si telle ou telle est pour lui importante ou sans intérêt. Il orientera son achat en conséquence. En dehors des points critiques mentionnés, tous ces produits font preuve de beaucoup d'homogénéité, surtout ceux fonctionnant sur Macintosh ou sous Windows. Les autres sont moins simples à utiliser, mais on gagne un temps précieux si on sait à l'avance ce qu'on doit y trouver.

Ces logiciels s'obtiennent soit sous forme de disquettes dans le commerce, soit sous forme de fichiers importés *(téléchargés)* par un réseau. On a déjà vu l'importance des fichiers **README** dans les répertoires des serveurs *ftp* d'Internet. Il en va de même avec les logiciels commerciaux, dans lesquels il faut avant tout chercher la présence d'un tel fichier : il contient de précieuses informations, qui ou bien viennent compléter celles de la documentation fournie, ou bien expliquent comment installer ce logiciel. On lit le fichier **README**, **LISEZ.MOI**, ou similaire, avec un éditeur ou, à la rigueur, avec la commande **MORE <** du DOS (cf. exercice 4, page 72 et note 8, p. 66).

Parfois ces logiciels s'installent par recopie pure et simple sur le disque dur des fichiers de la disquette ; cependant cette installation passe de plus en plus souvent par l'intermédiaire d'un utilitaire fourni avec le logiciel et appelé par exemple **INSTALL.EXE** ou **INSTALL.BAT**, car fréquemment les logiciels livrés sont *comprimés* et *compactés* : leur décompactage et leur décompression ne pourront se faire que grâce à cet utilitaire. Dans le cas des fichiers-réseau, ces opérations s'obtiennent avec les utilitaires universels dont on a parlé précédemment (un fichier *compacté* ou *archivé* contient un grand nombre de fichiers exploitables).

On redoublera de précautions si on installe un logiciel dit *protégé* (par protection logicielle) car toute fausse manœuvre pourrait lui être fatale (chap. XVII, p. 151). Heureusement, ce procédé semble être en voie de disparition (en 1995).

Avec chacun de ces logiciels, on doit lire le plus tôt possible et avec attention **la licence**, qui décrit les droits de l'utilisateur (on en parlera dans le dernier chapitre). Cette précaution est également nécessaire avec les logiciels acquis par le réseau. Noter que, dans ce dernier cas, le logiciel est soit gratuit, soit peu onéreux. Par contre, on ne disposera que d'une documentation réduite et parfois insuffisante pour parvenir à faire fonctionner correctement le logiciel (on peut alors explorer les fichiers **FAQ**, *frequently asked questions*, du répertoire d'où l'on a importé ce logiciel).

Chapitre XVI

L'INFORMATIQUE AU TRAVAIL

Peu de secteurs de l'activité humaine ont résisté à la poussée tentaculaire de l'informatique. On peut comparer celle-ci à un outil presque universel, améliorant dans de notables proportions le rendement de ceux qui l'utilisent.

Son succès s'explique en partie par l'économie de peine et de fatigue qu'elle entraîne dans maints travaux fastidieux, surtout dans ceux à caractère répétitif. Elle a apporté en ce sens un progrès comparable à celui dû à l'avènement des moteurs thermiques, puis électriques, à la fin du 19e siècle.

Cependant, comme on le verra bientôt, la principale raison de la poussée informatique n'est pas d'ordre philanthropique, mais avant tout économique, parce que l'informatisation d'un processus constitue presque toujours un investissement hautement rentable, s'il est opéré avec un minimum de compétence et de discernement.

Dans notre société en compétition généralisée, tout groupe qui refuserait les moyens apportés par l'informatique ne tarderait pas à se voir distancé par ses concurrents. C'est la raison qui a dû finalement emporter la décision de plus d'un réticent.

Dans ce chapitre, on fera un rapide survol des secteurs dans lesquels l'informatique s'est particulièrement implantée.

1 - LA PRODUCTION INDUSTRIELLE

On peut recenser trois formes principales d'intervention de l'informatique dans l'industrie : CAO, FAO et robotique.

a - La **CAO**, *conception assistée par ordinateur* (voir p. 122, chap. XIV, §6), a beaucoup modifié le travail des bureaux d'études. Le projet d'un nouveau produit comporte des phases de dessin industriel autrefois laborieuses. Les logiciels de CAO permettent d'assembler rapidement à l'écran, en respectant les conventions en usage dans la profession, des figures variées extraites de bibliothèques logicielles. Changement d'échelle et calcul des cotes deviennent automatiques. La vérification de l'assemblage des pièces entre elles est très rapide, les objets dessinés pouvant être déplacés, agrandis ou réorientés. Le gain de temps dans l'élaboration d'un projet est considérable. Dans un appel d'offres par exemple, la CAO permet de joindre au dossier de proposition des dessins très élaborés, malgré la brièveté du délai de réponse **(1)**.

La CAO s'est également emparée des cabinets d'architecte, comme on l'a mentionné dans le chapitre *Infographie II*. Elle s'infiltrera certainement bientôt chez les artisans, ceux du bâtiment, ceux du mobilier, etc, dès que son coût sera raisonnable. Actuellement, l'investissement souhaitable est important, tant en logiciel qu'en matériel (écran de grande taille, traceur de grand format, ...). Il est cependant possible de démarrer avec les périphériques ordinaires de la bureautique. L'effort de conversion nécessaire reste également un aspect non négligeable de la question.

La **DAO** est un terme parfois employé en lieu et place de la CAO lorsque la finalité du travail ne consiste pas dans une fabrication, mais se limite au dessin. Ainsi la cartographie utilise-t-elle amplement la DAO, quelle que soit l'échelle. L'informatisation du cadastre et du plan des villes est d'une utilité incontestable spécialement à cause de la facilité de mise à jour qu'elle entraîne.

b - La **FAO**, fabrication assistée par ordinateur, concerne l'étape venant normalement après la CAO. Elle consiste à commander les machines-outils par un programme informatique. Depuis plusieurs années, ces techniques ont conquis la mécanique, où les **machines numériques**, tours, fraiseuses, décolleteuses... sont largement implantées. Mais beaucoup d'autres branches de l'industrie se sont laissé gagner. Il suffit pour s'en convaincre de voir tailler les pièces d'un vêtement dans un rouleau de tissu par un robot de découpe laser : la rapidité, la dextérité de la machine et l'optimisation de ses coupes pour aboutir à un minimum de rebuts sont particulièrement saisissants.

---

(1) Il existe également une sorte de CAO à 3 dimensions, la *stéréolithographie* : un périphérique reproduit automatiquement la pièce dessinée. La matière est une résine photosensible se polymérisant tranche par tranche sous un faisceau laser mobile.

La **GFAO** (gestion de la FAO) va plus loin encore. Elle régit l'organisation du travail, l'**ordonnancement** de ses différentes phases, permet un approvisionnement rationnel des produits de base avec le minimum de stock, contrôle la qualité des produits finis, ... Poussée à l'extrême, elle aboutit au concept d'*usine flexible*, capable d'être remodelée en quelques jours pour fabriquer un produit nouveau. A un stade moins avancé, elle permettra de commander un produit de série totalement personnalisé (par exemple une voiture avec n'importe quelle option disponible). Sa fabrication, complètement informatisée, serait très rapide (quelques jours) et ne démarrerait qu'après passation de la commande. Un tel procédé ne peut être que plébiscité par la clientèle tout en diminuant pour le fabricant les stocks d'invendus.

c - La **robotique** couronne le tout. C'est une technique qui s'est surtout développée grâce à l'informatique, mais elle fait appel à bien d'autres disciplines, dont la mécanique de précision et la physique des capteurs. La *mécanique de précision* est la clé de voûte du développement des machines numériques et des *robots*, car elle leur donne la finesse d'action du bras ou de la main de l'homme. De tels mécanismes sont animés par des *moteurs pas à pas*, bien adaptés à l'informatique, car ils avancent – ou reculent – d'un petit angle donné à chaque impulsion envoyée par les circuits numériques.

La robotique exige en plus une grande maîtrise des *capteurs* afin de munir le robot du minimum de **perception** indispensable à son travail. Un robot n'est en effet pas rigidement programmé, il sait s'adapter à l'environnement, à la situation. C'est pour les appréhender qu'il possède des sens artificiels : toucher, radar, sonar et même *vision* plus ou moins élaborée. Certains d'entre eux seront bientôt dotées d'organes de vue supérieurs à l'œil humain, parce que capables de perception dans les trois dimensions. Le traitement de l'image reçue n'est pas une mince affaire et requiert des prouesses de la part de l'informatique associée.

Les robots sont en train de conquérir les postes de travail les plus mécanisables, les plus inconfortables, le travail à la chaîne par exemple. Certes un robot coûte cher, mais il faut bien admettre que, même sujet aux pannes, il est beaucoup plus fiable et *corvéable* qu'un ouvrier.

Si l'informatique peut être considérée comme bien implantée dans l'industrie, jusqu'ici la **production agricole** n'a pas été révolutionnée par elle. Bien sûr, les agriculteurs peuvent lui confier la gestion de leurs comptes, mais cela ne va pas loin. Plus sérieuse sera la mise au point de *systèmes experts* agricoles qui, à partir d'informations sur les marchés, les épizooties, les prévisions météo, seront capables de conseiller sur le type de culture à entreprendre, de prévoir les travaux en temps utile, le traitement des végétaux, l'épandage d'engrais ou l'irrigation ... afin de gérer au mieux les ressources pour le plus grand bénéfice de l'écologie.

2 - LE COMMERCE

Les répercussions de l'informatique sur le commerce paraissent moins profondes, moins structurelles, que sur la production. Elle permet tout de même aux grands magasins de **gérer** et de vendre beaucoup plus de marchandises à moindre coût et avec moins de personnel.

L'état des **stocks** est constamment tenu à jour et avec précision, ce qui permet de les réduire au strict minimum (leur coût est un point important dans le bilan financier d'une entreprise ou d'un commerce). Le strict minimum aboutit au concept de *juste à temps* ou de *flux tendu* : on commande le plus tard possible pour que, compte tenu des délais de livraison, le produit arrive juste à temps. Plus de stock chez le vendeur ou l'industriel : il est sur les camions.

Le **code-barre** est la condition quasi indispensable de l'informatisation du commerce. Ce procédé consiste à attribuer à chaque produit – au niveau du grossiste ou du fabricant – un numéro l'identifiant sans ambiguïté (voir l'annexe 3). Ce numéro, codé sous forme d'un jeu de barres, est imprimé sur tous les emballages du produit. Lors des réajustements de prix, il n'est plus nécessaire de réétiqueter les paquets, ni de fournir de nouvelles listes aux caissières. Le calculateur associé à la caisse lit le code-barre grâce à un capteur photo-électrique (crayon mobile à diodes émettrice et réceptrice ou CCD fixe), identifie le produit, le déduit du stock, consulte la liste de prix reçue par disquette ou par réseau, donne le prix réactualisé et l'imprime sur la note. Encore faut-il que le même prix ait été correctement affiché en rayon à l'intention du client (cet aspect du problème est à l'heure actuelle peu informatisé et parfois sujet à des défaillances).

Plus le point de vente est important et plus l'informatique peut y jouer un grand rôle. Les plus grands bénéficiaires en sont les sociétés de vente par correspondance, surtout quand on généralisera le *catalogue sur vidéo-disque* (catalogue animé et interactif !). Mais le petit commerce peut également en tirer parti : gestion précise des réserves, relevé journalier de l'activité, déclarations fiscales ... sont autant de tâches fastidieuses dont se joue l'informatique.

3 - LA BANQUE

Une grande partie du travail des employés de banque consistait, il y a peu, à enregistrer des mouvements de fonds et à tenir les comptes des clients. L'informatique a trouvé là très vite une application à sa mesure, en effectuant ces tâches de manière plus rapide et plus sûre que les meilleurs comptables. Les réseaux ont permis ensuite d'interconnecter entre eux les calculateurs des différentes agences, rendant la communication plus rapide au sein de l'organisme. Plus audacieux encore, des organismes différents se sont reliés les uns les autres par des grands réseaux échangeant un nouveau type de monnaie, la **monnaie électronique** ou *monétique* (sur ces réseaux, bien sûr, les informations sont *cryptées*).

Le pouvoir d'émettre une telle monnaie est parfois dévolu à la **carte bancaire**, support d'information permettant au système informatique de vérifier les droits du porteur *(code secret)* et de limiter la transaction à une valeur modeste (dans un intervalle de temps donné : jour, semaine, mois ...). De nouvelles cartes *(cartes à puce*, à mémoire) apporteront bientôt encore plus de possibilités tout en améliorant la confidentialité et la sécurité de la transaction.

Si pour l'instant, la vérification de l'identité du titulaire se borne à la reconnaissance d'un code numérique secret, on peut prévoir dans l'avenir le contrôle de sa voix, de son visage ou de sa signature. Le contrôle de la voix est déjà utilisé dans certains établissements pour vérifier les *droits d'accès* d'un demandeur ou ceux d'un visiteur à l'entrée de certains locaux *sensibles*. Cette escalade dans la recherche de preuves (appelées *signatures* au sens large) résulte de la compétition permanente entre l'ingéniosité des fraudeurs et celle des responsables de la sécurité des dépôts.

Le traitement des comptes bancaires ressemble à plus d'un titre à la manipulation d'une base de données avec accès multiples. Mais ce traitement possède des caractères spécifiques, qui rendent cette gestion parfois délicate : particulièrement cruciaux en effet sont les problèmes de demandes d'accès *simultanées* à un même compte ou celui de la panne survenant au cours d'une *transaction*.

Un nouveau type de banque commence à se développer : sans agence, ni guichet, elle n'est accessible que par télécommunication et l'informatique y fait la loi. Bien que très impersonnel, ce système bancaire est attractif pour la clientèle à cause de ses tarifs compétitifs provenant de la diminution de leurs frais, spécialement de ceux sur le personnel.

4 - LA REPARATION

Par ce terme, nous désignerons un secteur d'activité complètement opposé à celui de la production. Ce dernier tend en effet à produire au plus bas prix le plus de biens possible, tandis que celui de la réparation tend à prolonger leur durée, sans en augmenter le nombre. Ce secteur recouvre partiellement celui désigné parfois sous le nom de *services*.

Le travail impliqué est beaucoup plus diversifié que dans la production; jusqu'ici, il a beaucoup moins prêté le flanc à l'automatisation. Le réparateur doit traiter chacun des objets confiés de manière individuelle. Ici, pas de travail à la chaîne. A chaque phase, un raisonnement doit confronter le savoir-faire du *dépanneur* avec la situation, pour identifier les défauts et proposer les remèdes. On regroupera tous les spécialistes de ce genre sous le nom d'**experts**.

Pour arriver à reproduire le raisonnement de l'expert, les informaticiens ont dû développer l'**intelligence artificielle** **(2)**, nouvelle et passionnante discipline. En plus de la connaissance de l'informatique générale, de ses objets, de sa logique, des limites des langages machine, cette science exige l'étude des mécanismes du raisonnement ; on a dû approfondir nos connaissances sur la logique, la psychologie, la sociologie, la sémantique et même sur la genèse des langues humaines.

Le cerveau de l'homme ordinaire ne manipule guère les nombres, mais plutôt des images ou des **concepts** qui s'enchaînent les uns aux autres de manière apparemment *non déterministe* (un concept n'appelle pas forcément toujours le même). C'est par un cheminement complexe, incluant beaucoup d'essais et des retours en arrière, que la pensée parvient à une solution, et pas forcément à la meilleure. Pour reproduire ces raisonnements, il a fallu inventer de nouveaux langages, tels que le **Lisp** et le **Prolog** (on peut même classer dans cette catégorie le **Logo**, bien que d'ambitions plus modestes). Ils sont bien différents des langages décrits dans les chapitres IX et X.

Ces langages exploitent à l'extrême les notions de *structure* et de *récursivité* déjà présentes dans les meilleurs langages de programmation. Ces deux notions se conjuguent pour donner celle d'*arbre*, collection ou **liste** d'objets dont chacun peut devenir un **nœud** auquel se rattache une nouvelle collection dotée des mêmes propriétés. De tels arbres représentent

(2) Le domaine d'application de l'intelligence artificielle déborde largement celui de la réparation et s'étend en particulier à la robotique, à la théorie des jeux, à la reconnaissance de l'écriture ...

la **base de connaissances** des *systèmes experts*, logiciels conçus pour reproduire le travail des spécialistes. Le logiciel explore ces arbres verticalement, de la racine jusqu'aux branches extrêmes ou vice versa (contrairement aux SGBD qui explorent les bases de données de manière horizontale). C'est ainsi qu'est simulé le raisonnement : au passage, il privilégiera l'un des chemins comme étant le meilleur en fonction à la fois des critères de sélection retenus ou du but à atteindre, ainsi que de **règles déductives** d'association logique entre concepts *(moteurs d'inférences)*.

De tels systèmes sont déjà opérationnels, mais ils servent moins à remplacer l'expert qu'à le seconder et le rendre plus fiable. On voudra bien nous excuser d'assimiler la **médecine** à la réparation de la machine humaine, mais ce raccourci nous évitera d'ouvrir une section spéciale pour cette profession. C'est en effet dans ce domaine qu'à notre connaissance sont apparus les premiers systèmes experts. Ils en sont encore au stade expérimental. Le médecin qui en est doté en retire l'avantage de pouvoir confronter son propre diagnostic à celui de la machine – un confrère en quelque sorte – dont les connaissances seront celles d'un praticien peut-être pas génial, mais non soumis à des influences externes (fatigue, préoccupations, manque de temps ...) risquant de perturber la sérénité de son jugement. De plus, au moment du choix du remède, le médecin n'aura plus besoin de recourir à ses documents pour vérifier les propriétés des nouveaux médicaments, la mémoire de l'ordinateur étant là pour lui en présenter tout l'inventaire, avec leurs contre-indications et ses suggestions.

5 - LE SECTEUR TERTIAIRE

Dans le monde des *bureaux*, l'informatique n'a, semble-t-il, engendré qu'une mini-révolution. La *machine à écrire*, certes, a été reléguée aux archives. L'ordinateur est roi, les réseaux se développent, hiérarchie oblige... Grâce à eux, les *dirigeants* sont en prise directe avec leurs cadres et pianotent eux-mêmes leurs notes de services distribuées par *courrier électronique* sur réseau local. Les bases de données leur fournissent des informations aisément consultables et bien tenues à jour. Or on sait que la qualité de l'information est l'une des clés de la pertinence des décisions.

Les logiciels de **bureautique** ont fait décroître les délais de production des documents tout en améliorant leur qualité. Les tableurs permettent à des non-spécialistes de tenir une *comptabilité* honorable. On assurait que le règne de l'informatique avec ses réseaux et ses supports magnétiques devait faire chuter la consommation de papier ; la diminution ne paraît pas significative pour l'instant.

Dans un domaine voisin de la gestion, celui de la **statistique**, l'informatique s'est taillé la part du lion. Vouée aux traitements de masse à caractère répétitif, la statistique avait dû son essor à l'arrivée sur le marché des machines mécano-comptables au début du XXe siècle. La disponibilité actuelle de calculateurs puissants lui permet maintenant d'aller nettement plus loin que le simple constat des faits en élaborant des *modèles d'évolution* pour les phénomènes étudiés. Ces modèles autorisent ensuite une certaine prédiction de l'avenir *(prospective)*.

Nombre de **professions libérales** ont bénéficié de l'informatique. Des logiciels spécialisés sont disponibles pour certaines d'entre elles. Cependant, l'aide apportée jusqu'ici concerne plus leur aspect *gestion* que leur savoir-faire spécifique. Sauf pour l'architecture et la médecine (cf. §1 et 4), cette aide relève encore de la bureautique en attendant la contribution prévisible des systèmes experts.

Dans le monde des **loisirs**, l'informatique s'est implantée sans y apporter de révolution. Les organisateurs des manifestations sportives l'emploient pour le suivi des compétitions. Des concurrents y ont parfois recours, comme les grands *navigateurs*, pour le calcul de la route.

Les **transporteurs** ont déjà largement fait appel à l'informatique pour offrir à l'usager des voyages plus sûrs et plus rentables. La régulation des trains (aiguillages), celle des avions (tours de contrôle) en bénéficient déjà partiellement. L'enseignement du pilotage par *simulateurs* assure une meilleure formation avec moins de risques et à moindre coût. Côté commercial, planification du fret et réservation des places font appel aux réseaux pour assurer un taux de *remplissage* optimum. Les moyens disponibles devraient également bientôt bénéficier aux usagers pour leur plus grand agrément.

Les moyens audio-visuels utilisés par les *médias* sont particulièrement sensibles à l'informatique. Les professions impliquées ont d'ailleurs toujours été avides de nouveauté ou de progrès technique. Depuis plusieurs années, les **journaux** sont composés et imprimés par des machines informatisées. La **télévision** "consomme" beaucoup d'images élaborées ou modifiées par des calculateurs (cf. chapitre XIV). Cette tendance se développera encore, avec l'essor du *disque vidéo*, des procédés de *manipulation d'images* et de l'infographie en général. L'image gagne du terrain au détriment du parlé et de l'écrit. Sans doute bientôt, se posera avec acuité le problème de la crédibilité des images diffusées, tellement les procédés de manipulation vont devenir puissants, même sur des images censées être prises en direct (voir chap. XIV, page 125).

6 - LES ARTS ET LA CULTURE

En ce qui concerne l'**enseignement** – vecteur principal de la propagation de la culture – l'informatique s'est bien vite comportée comme un auxiliaire précieux : bien qu'encore peu répandu dans les classes faute sans doute de moyens financiers, l'ordinateur peut servir d'assistant, de répétiteur, voire d'examinateur. Il est employé pour compléter l'enseignement du maître et pour soumettre l'élève à une *auto-vérification* de ses connaissances. On trouve dans le commerce (à des prix abordables) des logiciels d'aide à l'apprentissage de diverses matières, langues, grammaire, mathématiques, ... La parution de véritables *encyclopédies* sur **disque optique** avec de nombreuses illustrations devrait contribuer à élargir le savoir et à exporter la culture loin des bibliothèques traditionnelles. Ce domaine est en plein développement, le matériel est à prix abordable, mais, en 1995, les productions de qualité sont encore rares ou chères (surtout celles en français).

Dans le secteur du **livre**, les avancées de l'informatique sont notables. Les logiciels de *traitement de texte* facilitent beaucoup le travail de l'écrivain. Le conseil du poète reste toujours vrai – *vingt fois sur le métier remettez votre ouvrage* [(3)]– mais ce travail devient aisé. Plus de *manuscrits* successifs, tout réside dans les mémoires du calculateur ; l'original de l'auteur (sur disquette) s'appelle un *compuscrit*. L'impression elle-même a subi une révolution. La **typographie** disparaît au profit de la *composition électronique*, plus rapide et moins coûteuse. Les documents tapés sous TTX sont *mis en page* à l'écran (grand format) avec des logiciels spécifiques (**PAO**, *publication assistée par ordinateur, desk-publishing)*, qui se chargent également de l'implantation des illustrations. Ces logiciels sont capables de piloter directement les *photocomposeuses* produisant le cliché définitif destiné à la reproduction. Tous ces facteurs devraient entraîner une explosion dans la production du livre et diminuer son coût.

Dans les **bibliothèques**, les SGBD vont devenir d'un grand secours. Grâce à eux, le lecteur retrouve un ouvrage même s'il n'en a que quelques notions et en a oublié titre, auteur, éditeur ... Ces notions seront traduites en **mots-clés**. Après tri du fichier de la bibliothèque par le SGBD en fonction de ces mots-clés, le demandeur obtiendra sur l'écran le résumé et les références des ouvrages cherchés. Le système lui indiquera la date de disponibilité des livres, après consultation du fichier des prêts (dont on espère qu'il restera confidentiel). Ces possibilités ne sont pas encore beaucoup exploitées, sauf dans les grandes bibliothèques et les *banques de données*, qui sont de véritables bibliothèques logicielles (c.-à-d. sans prêt d'ouvrages) gérées par un *serveur* relié à des réseaux. La difficulté principale qui freine l'informatisation des bibliothèques traditionnelles réside dans le volume de travail nécessaire à la *saisie* informatique des fichiers manuscrits existants. Mais le coût de ce travail diminue rapidement.

Les **arts graphiques** paraissent pour l'instant peu touchés, mis à part ceux à vocation commerciale (dessin animé, dessin publicitaire, télévision, cf. §5). Quelques artistes s'emploient cependant à trouver de nouvelles formes d'expression dans les ressources de l'informatique. L'équipement nécessaire (grand écran, imprimante couleur de haute résolution, logiciel de dessin haut de gamme) est encore trop cher pour que cette technique soit à la portée des nombreux peintres amateurs. A signaler quelques tentatives de constructeurs (Macintosh) pour mettre du matériel de leur marque à la disposition des intéressés (seulement dans quelques très grandes villes). Il y a certainement là un exemple à suivre, peut-être pour les associations ou les collectivités.

La **création musicale** par ordinateur est beaucoup plus répandue, pour des raisons peut-être discernables. L'investissement nécessaire est certainement moins élevé ; il est d'autre part de l'ordre de grandeur des sacrifices financiers que plus d'un amateur de musique avait déjà consenti pour satisfaire sa passion. Ajoutons que l'exploitation commerciale des œuvres produites est plus rentable que celle des œuvres picturales : à cause des facilités de reproduction, toute œuvre acceptée par une maison de disques est source de revenus immédiats. Enfin, il faut noter que cette création et sa consommation concernent une population plus jeune que celle du domaine des arts graphiques et que ceci explique peut-être cela.

On connaissait déjà les **synthétiseurs** et leurs variantes (orgues électroniques par exemple). Il s'agit d'appareils électroniques, pilotés par un clavier du type piano ou orgue : leurs haut-parleurs émettent des sons sous l'action de signaux amplifiés issus d'oscillateurs. Ces instruments sont dotés de sonorités nouvelles qualifiées parfois de *spatiales* ou même de *psychédéliques*, dont la *musique de variété* fait largement usage. Quelques compositeurs contemporains ont réussi à tirer de ces instruments des œuvres remarquables.

Récemment est apparu l'**échantillonneur** (en anglais *sampler)*, qui est avant tout un **numériseur**, capable de prélever à une cadence honnête (44 à 48000 fois par seconde) un échantillon sur un signal issu d'un microphone ou d'un synthétiseur, de le numériser et de le mettre en mémoire. Cette forme de conservation musicale est exactement celle des disques numériques grand public *(compact, laser, ...)*,

(3) Boileau, *l'Art poétique*.

dont on connaît l'extraordinaire qualité. Cette musique pourra être reprise avec un logiciel de conception musicale, pour être transformée (en hauteur, timbre, *tempo* ...) ou orchestrée. Il faut munir le calculateur d'une carte spéciale, raccordée aux synthétiseurs, aux échantillonneurs et à divers autres par un bus de type série à 32 kbauds appelé *interface MIDI*, satisfaisant à la norme de même nom *(musical instruments digital interface)* et capable de réunir jusqu'à 16 appareils. Ces logiciels musicaux sont même conçus pour **enseigner** la musique. C'est Atari qui a joué le rôle de pionnier en ce domaine. Sa place y est encore prépondérante, mais elle partage le marché avec des concurrents comme Apple, Amiga et IBM.

Le développement de ces techniques a introduit la notion de **multimédia**. Ce terme signifie que l'information est maintenant transmise par plusieurs vecteurs : texte, son et image (fixe ou animée). Ces deux derniers vecteurs sont très gourmands en mémoire ; ils exigent donc en pratique le disque optique (CD-ROM). Pour les lire, il faut, en plus de l'ordinateur, un lecteur optique et une carte vidéo de qualité. Pratiquement, ce matériel implique également la présence d'un bus local rapide (plus rapide que le bus ISA), indispensable pour les images animées.

Le multimédia peut, à court terme, apporter de profonds changements dans la diffusion des arts et de la connaissance. Le disque optique possède beaucoup d'atouts sur le livre : légèreté, capacité, facilité de consultation... Ses principaux défauts résident dans le prix du matériel nécessaire à sa lecture et dans la lourdeur de sa mise en œuvre. Il ne fait guère de doute que le rapport qualité sur prix de ces vecteurs évoluera rapidement en faveur du disque optique.

7 - REMARQUES

A propos de l'introduction de l'informatique dans l'art, on peut se demander si elle sera capable de **création** et on a souvent répondu par la négative à ce genre de question : le calculateur ne pourrait produire que ce pour quoi il est programmé. C'est oublier un peu vite l'intervention du hasard. Si la création peut se ramener à la conjugaison du hasard et de l'esprit critique, la création par ordinateur pourrait bien être pour demain.

Au plan culturel, l'emprise de l'informatique de type personnel aura pour conséquence générale le renforcement de la personnalité. L'utilisateur disposera de moyens extrêmement serviles et efficaces qui lui permettront de faire aboutir ses idées ou ses projets sans aide ou avec beaucoup moins d'aide extérieure. L'informatique, pour ceux qui sauront sans servir – et s'en servir avant les autres – sera l'amplificateur de leur dynamisme. En contrepartie, des aspects négatifs se feront vite ressentir, par exemple celui d'un excès d'individualisme ou celui de la solitude du créateur devant sa machine pour longtemps encore dénuée de tout sens critique.

En attendant, l'informatisation généralisée ne se fait pas sans engendrer des conflits chez ceux qui la subissent. Le passage à l'informatique est beaucoup mieux vécu dans le monde de la technique que dans celui de l'administration, où de nombreux mécontentements ont été soulevés. Il est vrai que certains formulaires sont devenus difficiles à comprendre ou à remplir après une telle mutation. On y retrouve parfois l'obligation de répondre à des questionnaires par des nombres au format rigide, avec même des zéros en tête ... On se croirait revenu 30 ans en arrière ! Les codes secrets continuent à être numériques, donc destinés à être oubliés.

Cette situation a souvent donné une image négative de l'informatique, alors qu'elle ne provient que d'un manque de compétence chez les programmeurs ou de la volonté délibérée chez certains responsables de ne pas se mettre au service du public. On est ici parfois bien loin de la *convivialité*. Et pourtant jamais ce mot n'a été autant employé que depuis la révolution informatique.

Chapitre XVII

INFORMATIQUE ET SOCIETE

Darwin a prouvé que l'évolution du monde vivant dépendait de sa faculté d'adaptation aux modifications de son environnement ainsi qu'à la compétition entre les espèces. On n'oserait assimiler ces mécanismes millénaires à l'irruption de l'informatique dans la société du XXe siècle finissant. Mais, toutes proportions gardées, il s'agit bien de modifications de notre environnement technique, culturel et social, d'adaptation à ces modifications et de lutte entre clans pour l'appropriation de ces nouvelles ressources.

Sous l'emprise de l'informatique, des métiers disparaissent, d'autres apparaissent, des pouvoirs nouveaux se forgent, des conflits surgissent, des coalitions s'élaborent... Tel est en gros le schéma qui nous guidera dans ce dernier chapitre pour esquisser le tableau des bouleversements prévisibles et ceux déjà amorcés dans les sociétés industrialisées.

On s'apercevra finalement que le bilan n'est pas aussi tragique que d'aucuns le disent. Il est même porteur d'un espoir pour l'ensemble de la planète, dans la mesure où les détenteurs des pouvoirs politiques et économiques feront preuve de la sagesse indispensable à une répartition harmonieuse des richesses créées par l'informatique.

1 - LES PERTES D'EMPLOI

Dans le chapitre XVI, nous avons relaté l'intrusion de l'informatique dans différents secteurs de l'activité humaine. On a constaté sa pénétration de plus en plus profonde dans la grande industrie et les bouleversements provoqués dans la **production de masse**. Sont en train de disparaître les emplois les moins qualifiés, tels les travaux à la chaîne, mais également ceux ne requérant qu'une compétence accessible aux **robots** (tri du courrier, nettoyage des locaux, gardiennage,...). De même, le commerce n'a pas terminé sa reconversion : la vente par correspondance ne peut que se développer sous l'effet de la banalisation des réseaux et du disque vidéo ; d'autre part, le secteur des "grandes surfaces" lui-même n'a pas fini de supprimer des emplois (en particulier celui de caissière).

C'est en partie pour cela que l'avènement de l'informatique a été qualifié de *révolution industrielle*, la seconde ou la troisième selon les auteurs. On a dit qu'il était comparable à l'irruption du machinisme au 19e siècle à la suite des inventions remarquables de cette époque. Tout le monde a entendu parler des révoltes ouvrières contre la machine à coudre de Thimonnier ou contre le métier à tisser de Jacquard. Les machines ne s'en sont pas moins imposées.

Si les inventeurs sont souvent motivés par le souci d'épargner la peine des travailleurs, l'invention, une fois dans les mains des technocrates, ne se développe que dans la mesure où elle est *rentable*. Or l'informatique l'est sans conteste dans la production industrielle. Le robot dépasse en rendement le manœuvre, car il travaille sans relâche, ni congé, ni protection sociale. Il remplace même à lui seul plusieurs employés. Il n'est pas étonnant qu'il supprime beaucoup de postes.

On remarquera que le Japon, passant pour disposer d'une main-d'œuvre adroite, fidèle et pas trop exigeante, est pourtant le pays le plus robotisé au monde. Or son taux de chômage est l'un des plus faibles qui soient. Ce qui porte à croire qu'informatique et chômage ne sont pas forcément liés. De même, plusieurs pays d'Extrême-Orient accusent un taux de croissance élevé, avec l'aide de l'informatique. L'objet de ce livre n'est pas d'étudier l'économie de ces pays, mais ces exemples devraient inciter à ne pas porter un jugement hâtif sur la relation entre informatique et sous-emploi.

On a vu que le secteur de la **réparation** n'est pour l'instant pas trop menacé, sauf si la production de masse à bas prix par les robots ne lui enlève toute rentabilité. Les *systèmes experts* devraient lui donner une vitalité nouvelle et une fiabilité accrue. Allongeant la durée des biens périssables, cet essor serait en harmonie avec la prise de conscience par certains de la nécessité d'épargner les *matières premières* (Club de Rome) et de restreindre la *production des déchets* (écologistes). Si électeurs et Etats décident d'aller dans ce sens, le secteur de la réparation, *assisté* par les systèmes experts, a de beaux jours devant lui et constituerait un excellent gisement d'emplois.

Du côté du bureau, les effets seront plus ravageurs. Les emplois de faible compétence y sont nombreux et disparaîtront à terme. Il est sûr que l'employé qui ne veut faire preuve ni d'initiative, ni d'imagination, ni de dynamisme – ni d'humanisme en quelque sorte – se condamne lui-même dans la compétition avec la bureautique. Seuls subsisteront les emplois requérant compétence, sens du service et maîtrise des techniques nouvelles.

La perte de ces emplois traditionnels sera en partie compensée par la croissance en nombre des **métiers** de l'informatique. Il n'y aurait compétition entre homme et machine que dans le domaine de la production. Ceci implique des reconversions d'activité et l'obligation pour beaucoup de *glisser* vers des emplois plus nobles, la machine accaparant les tâches subalternes. En somme, des humains de plus en plus compétents géreront un parc de plus en plus étendu de robots. L'un des freins à cette évolution pourrait venir de notre difficulté à nous adapter aux nouvelles techniques.

Le domaine des professions de l'informatique est suffisamment diversifié pour qu'on le subdivise en plusieurs secteurs. Il offre un éventail de compétences relativement large.

2 - LA PRODUCTION DU MATERIEL INFORMATIQUE

Dans ce secteur, on distinguera encore plusieurs catégories :

a - Tout d'abord l'industrie de la **micro-électronique**, axée sur la fabrication des *circuits intégrés* (*processeurs*, mémoires...). Seules des usines robotisées produisant en masse permettent de fabriquer des *puces* à faible prix malgré les lourds investissements en infrastructure, *salles blanches*, robots et recherche-développement. La *recherche* sert à mettre au point des circuits plus denses et plus performants, avec des matériaux nouveaux (AsGa, éléments II-VI), des techniques nouvelles (optoélectronique, supraconductivité), ou des concepts nouveaux (architecture parallèle, réseaux de neurones, logique floue...). Les sociétés impliquées dans cette production sont devenues peu nombreuses et leur implantation géographique est des plus restreintes : USA (spécialement dans la *Silicon Valley* en Californie), Japon et, à un moindre degré, Europe. On peut citer dans le domaine de la production européenne les sociétés Philips (aux nombreuses filiales), SGS (Italie) associée à Thomson (France), Siemens (Allemagne) associée à Nixdorf (Suède)... Cette production étant jugée vitale par les Etats, des plans de R&D associant plusieurs partenaires sont parfois impulsés par les pouvoirs publics (comme les programmes JESSI et ESPRIT en Europe)**(1)**.

b - Les fabricants d'**ordinateurs** sont un peu plus dispersés. Leur travail consiste à réaliser les cartes de circuits imprimés, à y souder les composants et à *assembler* les éléments de la machine. Le dernier point est peu automatisable et ne nécessite pas de compétences "pointues". Aussi les pays disposant de main-d'œuvre soigneuse et peu chère conquièrent chaque année de nouvelles parts du marché. La grande majorité des ordinateurs est maintenant fabriquée peu ou prou en Extrême-Orient, même sous des marques occidentales. Cependant, à moins de vouloir se limiter à des copies serviles, les maisons mères doivent entretenir des bureaux d'études et même des équipes de chercheurs. A cause de sa compétence séculaire, jusque au début des années 90, IBM dominait le marché mondial (30% des ordinateurs produits), mais Compaq, Dell, HP et autres sont devenus en peu de temps des rivaux pour *le géant*. En Europe, on compte surtout Olivetti (Italie), Bull (société française, plutôt orientée vers le monde industriel et Unix)... Beaucoup de petites sociétés, hélas souvent éphémères, montent des ordinateurs. Il ne faut pas oublier l'existence de puissantes filiales de firmes US, dont IBM et surtout Hewlett-Packard, qui contribue à faire de la France un grand exportateur d'ordinateurs.

Il y a une vingtaine d'années, le gouvernement français avait élaboré un *Plan Calcul*, visant à doter le pays de l'indépendance en matière d'ordinateurs. Ce plan n'a pas obtenu de brillants résultats. Il est vrai qu'il était surtout impulsé par les militaires et visait les gros calculateurs au moment où se concoctaient dans les laboratoires les ingrédients qui devaient aboutir à l'explosion de l'informatique par la prolifération des petits calculateurs.

c - Toujours dans le domaine de l'industrie informatique, il faut mentionner les fabricants de **périphériques**, à la pointe du progrès dans leur spécialité, mécanique de précision, dépôts magnétiques, photosensibilisation (disques durs, imprimantes, ...) en grande partie extrême-orientaux. Saluons au passage la prestation de la firme française Benson, spécialisée dans les tables traçantes haut de gamme, celle du complexe NEC-Olivetti (Italie) fabricant d'imprimantes à jet d'encre, ainsi que la contribution de Philips (à l'origine néerlandaise), dont les chercheurs ont élaboré plus d'un procédé, surtout dans le domaine des enregistrements magnétique et optique. N'oublions pas non plus les concepteurs d'automates et de robots (recherche, bureaux d'études, conception de circuits spéciaux – les *ASIC* –, mécanique fine, ...).

d - Restent à citer les fabricants de matériaux **consommables**, disquettes, cartouches laser, rubans encreurs. Ils semblent mieux répartis sur la planète que les précédents, peut-être pour des raisons de moindre technicité et de proximité de la clientèle. Il arrive qu'on assiste à l'éclosion subite d'activités nouvelles répondant à une demande précise du marché. Ainsi, vers 1990, il est apparu de petites sociétés *recyclant* les cartouches d'imprimantes laser. Ces sociétés se doivent cependant de rester attentives à l'évolution des techniques, leur dépendance envers un produit unique leur conférant une fragilité extrême.

(1) Tous les grands pays soutiennent leur industrie informatique, par exemple en finançant partiellement la recherche fondamentale en amont.

3 - LA PRODUCTION DU LOGICIEL

Les grands **logiciels** de bureautique, de dessin, de communication, de programmation, ... sont issus de sociétés de taille très inférieure à celle des producteurs de matériel. L'investissement est beaucoup moindre et n'implique guère que matière grise, savoir-faire et stations de travail. On appelle ces sociétés des *éditeurs* de logiciels. Leur implantation n'est guère mieux distribuée que celle de la construction, les USA se taillant la part du lion. La France y occupe une place honorable, mais, curieusement, Extrême-Orient et même Japon sont, pour l'instant, très peu présents sur ce créneau (sauf en ce qui concerne le secteur très rentable des jeux informatiques où le Japon est meneur). C'est Microsoft qui, en quelques années, est devenu le premier producteur de logiciel au monde, après avoir livré une guerre sans merci à ses rivaux, Borland, Lotus, ... et même IBM.

Une notable partie du logiciel est due à de petites sociétés, voire à des **particuliers**. La commercialisation de leur produit est difficile. Ou bien ces auteurs négocient la cession de leur œuvre à une grosse société d'édition, moyennant un contrat plus ou moins avantageux ; ou bien, aux USA du moins, ils utilisent le **shareware**. Ce procédé consiste à autoriser sans réserve (et même à encourager) la copie de leurs œuvres pour les diffuser au maximum. Une fois en service, ces copies affichent à l'écran un message demandant à l'utilisateur, s'il est satisfait et honnête, d'envoyer à l'auteur une somme modique pour lui permettre de continuer à créer de nouveaux logiciels. Aux USA, ce procédé donne de bons résultats.

Du côté de la *consommation*, déjà des sociétés se tournent vers la production d'œuvres plus ou moins culturelles du type *multimédia* sur disque optique. On parle d'*édition électronique* et les éditeurs classiques vont se convertir progressivement à ces techniques. Les enjeux sont très importants. Aux USA en 1995, ce secteur est le premier des créateurs d'emplois.

4 - LE COMMERCE DE L'INFORMATIQUE

Sa répartition est fonction du marché et l'Europe y figure cette fois-ci en bonne place, aux côtés des deux grands, USA et Japon.

Au sommet de la pyramide, on trouve les constructeurs et les *éditeurs*, qui ne vendent que rarement aux utilisateurs. Des intermédiaires, allant des *grossistes* aux plus modestes *boutiques*, diffusent à la fois matériel et logiciel ; souvent ils conseillent l'acheteur, parfois avec compétence, sur la configuration la plus adaptée à ses souhaits. Ce commerce est difficile car la concurrence est acharnée. La *compétence* est de règle si l'on veut s'y faire une place *durable* au soleil.

Le lecteur intéressé par l'informatique aura très vite affaire à ces *professionnels*. Il sera surpris par quelques aspects négatifs de ce commerce : d'une part la quasi-impossibilité d'être certain à l'avance – à moins de passer par une SSII – que les produits achetés satisferont son attente. Il lui faudra prendre auparavant beaucoup de conseils. Les *démonstrations* éventuelles du vendeur ne concernent que des problèmes connus de lui, rarement ceux du client. Le deuxième point noir consiste dans la **documentation**. Elle est à la fois volumineuse et en général fort médiocre. Rédigée sans souci pédagogique, souvent à la va-vite (pour *sortir* au plus tôt le produit sur le marché), parfois en anglais ou très mal traduite, elle en décevra plus d'un **(2)**.

Un autre désagrément attend notre candidat dans ses emplettes informatiques : c'est le **prix élevé** des logiciels. Bien souvent, on oublie d'en tenir compte dans le budget établi en prévision d'un équipement. Il faut pourtant savoir qu'à l'heure actuelle, le logiciel est plus onéreux que le matériel, à moins de se contenter de peu, c'est-à-dire des logiciels en libre service *(shareware*), de ceux tombés dans le domaine public *(freeware* aux USA) ou de ceux, de plus en plus nombreux, que des concepteurs idéalistes mettent à la disposition de tous sur le réseau Internet. Ces logiciels gratuits ou bon marché ont un inconvénient : les revues d'informatique les boudent et il est difficile de s'en faire une opinion (sauf par le bouche-à-oreille).

La **formation** à l'informatique par des organismes spécialisés est bien préférable à un auto-apprentissage avec les documents de piètre qualité habituellement fournis. Encore vaut-il mieux, à cause de la brièveté des stages proposés, ne les suivre qu'après un contact poussé avec l'objet de la matière enseignée. Le prix généralement élevé de ces formations les rapprochent plus du commerce de l'informatique que de son enseignement. Elles ne sont pour la plupart guère abordables au particulier, sinon par le biais de la *formation continue* des salariés d'entreprises.

(2) Beaucoup d'informaticiens préfèrent le texte en anglais, la traduction française étant souvent réalisée à l'étranger, ou bien en France, mais par des non-spécialistes.

5 - LES SOCIETES DE SERVICES EN INFORMATIQUE

Ces sociétés, en abrégé **SSII** ou **SSCI**, rassemblent des informaticiens aptes à mener une étude complète de problèmes posés par des clients et solubles par l'informatique. Rompues aux besoins de l'industrie et des administrations, les SSII peuvent proposer, puis mettre en service des installations *clés en mains*, matériel et logiciel inclus. Bien qu'utilisant au maximum les produits du commerce, leurs spécialistes sauront développer les programmes nécessaires, si les logiciels commerciaux font défaut ou sont insuffisants. Ces sociétés peuvent assurer la formation du personnel utilisateur. Enfin, grâce à leur matériel propre, elles sont également capables de résoudre des problèmes ponctuels purement intellectuels, c.-à-d. sans fourniture d'équipement.

Ce genre d'activité reste très proche des utilisateurs potentiels, aussi est-il plutôt bien disséminé dans les pays industrialisés. Cependant, des accords, rachats, prises de participation,... ont transformé certaines d'entre elles en multinationales. La France est bien placée dans ce domaine et les plus puissantes de ses SSII comptent parmi les premières mondiales (par exemple Cap-Gémini-Sogéti) ; c'est un secteur économique particulièrement dynamique.

6 - LES INFORMATICIENS DE TERRAIN

On appellera ainsi tous les autres professionnels de l'informatique. Certains d'entre eux, à vrai dire peu nombreux, ont le statut d'artisan ou de **profession libérale** : ils travaillent alors à peu près comme une SSII, mais à échelle bien plus modeste (programmation, dessin, CAO, comptabilité, frappe de documents, formation, assistance diverse...). Avec la banalisation des réseaux, de tels emplois devraient se développer. Caractérisés par un temps de réponse très court du fait de leur indépendance, en prise directe avec leurs clients tout en travaillant à domicile, ces informaticiens *libéraux* sont appelés à rendre de grands services aux PME, aux professions libérales et aux particuliers. Encore faut-il que les pouvoirs publics leur accordent l'attention qu'ils méritent.

Pour l'instant, la grande majorité des informaticiens de terrain sont **salariés** des gros consommateurs d'informatique, industriels, entreprises, administrations. Un organisme possédant une installation de calcul collective ou *a fortiori* une de grande puissance, ne pourra pas se passer d'une telle équipe de spécialistes. On y rencontrera les postes suivants : responsable du groupe, chef de projet, ingénieur système, analystes, programmeurs et – de moins en moins souvent – opérateurs.

L'**ingénieur système** est responsable du matériel et de son système d'exploitation. Il connaît bien celui-ci, ainsi que son langage machine ; il programme les *utilitaires*, élabore les fichiers de commandes *(fichiers batch)* nécessaires à l'exploitation du matériel et des réseaux. L'**analyste** étudie les problèmes posés et en donne une solution schématique, par exemple sous forme d'organigrammes, que le **programmeur** traduira en instructions dans le langage fixé.

Dans les usines automatisées, des équipes de haute compétence veillent sur les **robots** et assurent leur *maintenance*. Les ingénieurs qui les dirigent ont tendance à devenir des généralistes, le parc de machines gérées mettant en jeu des disciplines de plus en plus nombreuses.

Cependant, nombre d'industriels préfèrent, pour des raisons d'économie, se passer de telles équipes et confier leurs problèmes à des SSII. Le choix entre les deux solutions n'est pas des plus simples. Le problème majeur est celui de la **maintenance** de l'informatique à long terme. Il ne sera pas facile de faire modifier après quelques années un problème évolutif temporairement résolu, les auteurs de l'étude initiale risquant fort d'avoir changé de maison. En effet, chez les informaticiens et surtout dans les SSII, la **mobilité** *(turn-over)* reste élevée. Elle est causée par un déficit en spécialistes expérimentés et hautement qualifiés. Cet état de fait milite plutôt pour des équipes bien motivées et suffisamment intégrées à la société qui les emploie, mais pourrait également jouer en faveur des indépendants dynamiques et sérieux.

7 - LA DISSEMINATION DU TRAVAIL

Dans les sections précédentes, le lecteur a pu se rendre compte de l'impulsion que pourrait donner l'informatique à nombre de professions indépendantes : nous avons cité les techniciens de la réparation, ceux du recyclage et les informaticiens libéraux. Si l'on réfléchit sur le développement inéluctable des *réseaux*, on ne peut que prédire la déconcentration des lieux du travail. En effet, à quoi bon perdre 3 ou 4 heures par jour en *transport* quand on peut travailler chez soi avec les mêmes moyens et la même efficacité qu'en usine ? Certes, l'aspect *collectif* de la plupart des travaux ne disparaîtra pas pour autant, des rencontres

entre collaborateurs resteront nécessaires, mais l'obligation systématique d'être présent à l'usine 8 heures par jour et 5 jours par semaine devrait vite céder le pas à l'**horaire flexible** et à la réunion occasionnelle. Les travailleurs pourront ainsi habiter plus loin de leur *centre* de travail comme le feront certainement les informaticiens indépendants. On évoluerait ainsi vers une meilleure utilisation du territoire et une meilleure répartition des richesses.

De même, les *réunions* entre dirigeants d'une même société pourront s'avérer moins utiles, eu égard au temps perdu dans les déplacements, quand on se sera aperçu que la mise à jour quotidienne d'une base de données est le meilleur des comptes-rendus. Avec les moyens offerts par la téléphonie numérique, en particulier la télé- et la vidéo-conférence, le désir du PDG de sentir qu'il tient bien en mains tous ses chefs de division pourrait être satisfait sans réunions *physiques* de moins en moins justifiables. Ainsi les succursales ne seraient-elles plus obligées d'être aussi proches du siège social.

L'essor des réseaux informatiques devrait donc rendre beaucoup moins aigu le problème des **transports**, qu'il s'agisse des transhumances quotidiennes entre domicile et usine ou du remplissage des avions et des trains rapides vers la capitale. Si le temps perdu est récupéré en heures *ouvrées*, la productivité devrait s'en ressentir, ainsi que les économies en carburant et les pollutions sonore ou chimique dues aux moteurs thermiques.

A plus long terme, c'est l'**enseignement** lui-même qui pourrait être décentralisé. Le disque optique apportera la base des connaissances nécessaires, l'enseignant les explications. Le réseau reliant le maître à toute la classe évitera l'isolement dans lequel se trouve plongé actuellement l'élève d'un cours par correspondance.

Meilleure utilisation du territoire, revitalisation des campagnes, diminution des transports, de leurs pollutions et de leurs dangers, les perspectives de l'informatique ne sont donc pas aussi sombres qu'on a pu parfois les décrire.

8 - LA PROTECTION DU DROIT D'AUTEUR

Pour tirer de leur travail une juste récompense, les créateurs de programmes distribués en *shareware* ont imaginé un moyen astucieux (§3, p. 149) reposant sur l'honnêteté des utilisateurs. Les gros *éditeurs* de logiciels ne se comportent pas de la même manière avec leurs clients. Ils interdisent formellement :

– la copie de leur produit (toujours vendu sous forme de disquettes) à deux exceptions près : son installation sur disque (ou sur disquette-système) et une copie de secours,

– son prêt ou sa cession à un tiers à quel que titre que ce soit,

– la modification du logiciel livré.

Ils considèrent d'ailleurs que le logiciel vendu reste la propriété de l'éditeur et non celle de l'acheteur.

Ces clauses sont gênantes dans les entreprises où l'utilisation collective des moyens est la règle. On y avait salué avec soulagement l'arrivée des calculateurs personnels et des réseaux locaux. L'interdiction du partage des logiciels oblige les employés à revenir au calculateur commun, ce qui n'est pas ressenti comme allant dans le sens du progrès.

Certains *éditeurs* tolèrent que leur logiciel soit utilisé sur plusieurs postes, si **un seul** est opérationnel à un instant donné. Pour garantir cette clause, il leur arrive de *crypter* les disquettes fournies de façon à en rendre impossible plus d'une copie. On copie ce logiciel sur le disque dur, mais la disquette est alors *désactivée*. Il sera par la suite possible de *désinstaller* le logiciel du disque, ce qui réactivera la disquette et lui permettra d'exporter les programmes vers un autre calculateur **(3)**. Le procédé serait correct s'il n'y avait aucun risque de défaillance au cours des cycles d'installation-désinstallation, toujours longs et malaisés. Sous la pression de la clientèle, ce procédé est en voie de disparition.

Un procédé entraînant moins de déboires consiste à laisser le client – ou son groupe – copier sans restrictions le logiciel vendu, celui-ci ne pouvant fonctionner qu'avec une **clé** matérielle *(dongle, jeton, bouchon)* connectée à l'emplacement *ad hoc* (généralement la prise du câble d'imprimante). Ainsi, à un moment donné, seul le titulaire de la clé peut utiliser un tel logiciel. L'inconvénient de ce procédé réside dans l'empilement hasardeux des clés les unes derrière les autres quand on possède plusieurs logiciels de ce genre. Il vaudrait mieux que le connecteur nécessaire soit non pas celui de l'imprimante, mais un connecteur réservé à cet usage et situé en face avant. Mieux encore, le logiciel pourrait être protégé par *clé* en phase de partage, puis par *cryptage* lorsqu'on désire le conserver pour soi tout seul.

(3) La technique de cryptage repose parfois sur l'emploi de fichiers DOS d'attribut *caché*, parfois sur des écritures directes sur disque, ne faisant pas appel au système d'exploitation.

9 - LA LEGISLATION

Juristes et législateurs se sont penchés sur les conflits provenant de l'usage de l'informatique. Il en est résulté dans nombre de pays lois et décrets réglementant son emploi et que les utilisateurs doivent connaître. En France, les textes ci-après s'y rapportent.

La loi du 6 janvier 1978, appelée *informatique et libertés*, réglemente la constitution des **fichiers nominatifs**, i.e. de ceux qui permettent d'**identifier** les personnes *fichées* **(4)** (par *fichier*, il faut entendre ce que les informaticiens appellent *bases de données*). Une commission nationale (la CNIL) a un droit de regard sur eux; ils doivent faire l'objet d'une déclaration préalable auprès d'elle. Cette commission tient à la disposition du public la liste des fichiers déclarés, avec quelques précisions sur chacun d'eux (finalité, utilisateur, type de traitement ...). Un individu peut **parfois** s'opposer à *sa mise en fiche* ; il a en général le droit de se faire communiquer (à titre payant) les renseignements figurant à son égard et d'obtenir leur rectification s'ils sont faux.

Les fichiers relatifs à la *sûreté de l'Etat* et à *la sécurité publique* échappent à ce droit d'accès **(5)**, mais pas au contrôle de la CNIL. La loi ne fait que limiter un peu leur poids devant les tribunaux. D'autre part, beaucoup de sociétés cèdent ou vendent à d'autres (non concurrentes) des fichiers d'adresses en leur possession (clients par exemple). Ces fichiers permettent un démarchage intensif. Il est permis de s'opposer à cette cession, mais il faut le faire explicitement.

L'autre source de conflits que la loi a tenté d'arbitrer concerne le **droit d'auteur**. Il est admis – en principe – que tout travailleur reçoive une juste rétribution pour son ouvrage. Quand le produit relève du domaine de l'invention, la rétribution implique une participation aux bénéfices réalisés grâce à elle. D'où la législation sur les *brevets* ou celle sur la *création* artistique et musicale.

On sait qu'en ce qui concerne la musique, il est interdit de tirer bénéfice d'une œuvre sans payer une redevance à qui de droit (auteur ou société le représentant), mais qu'il est licite de la copier pour un usage personnel ou familial. La **loi du 3 juillet 1985** réglementant l'usage du logiciel est bien plus **restrictive** que celles sur les brevets ou la musique : le logiciel acquis auprès d'un vendeur ne peut être ni modifié, ni copié, même pas pour un usage personnel (à l'exclusion d'une copie de secours). Cette loi est tout à fait conforme aux vœux des *éditeurs* tels qu'ils sont exposés au début de la section 8.

Sur un autre sujet, la **loi Godfrain** du 5 janvier 1988 condamne la malveillance et la pénétration frauduleuse sur un système auquel on n'a pas accès ou auquel on accède en dépassant ses droits (voir également le code pénal, article 232).

Une **directive du Parlement européen** sur le droit d'auteur (11 juillet 1990) stipule que tout acheteur de logiciel peut le modifier ou en réutiliser certaines parties dans ses propres programmes. En outre, les bibliothèques publiques seraient autorisées à prêter des logiciels. Ce texte est en contradiction avec la loi française ; le débat sur le sujet n'est donc pas clos.

10 - LA LOI DE LA JUNGLE

Beaucoup d'informaticiens ont innocemment pratiqué, pour leur propre usage, des copies de logiciels commerciaux. D'autres le font sciemment, alléguant le prix exagéré des logiciels**(6)**. Les éditeurs répliquent que la cherté de leurs produits est due au manque à gagner causé par les multiples copies illégales, qu'ils qualifient de *pirates*. Il est vrai qu'il est arrivé (via un pays étranger à la législation plus souple) que de telles copies puissent être revendues.

Des *éditeurs*, groupés en de puissantes associations, ont obtenu la promulgation de la loi de 85. Forts de leur droit, ils ont déclenché des contrôles policiers dans des entreprises et des administrations, puis engagé des poursuites judiciaires contre celles-ci après constat en leur sein de copies illicites. Il semble qu'en général une "transaction" ait permis à ces organismes d'obtenir la suspension des poursuites. Tout informaticien, même amateur, doit savoir qu'il n'est pas à l'abri d'une telle *perquisition policière*, que les inspecteurs sont des spécialistes **(7)** et que les peines encourues sont sans rapport avec le bénéfice retiré de l'emploi de ces logiciels. Les commerçants qui céderaient des logiciels (souvent des systèmes d'exploitation) avec des machines sont répréhensibles et ces éditeurs demandent à leurs clients de les dénoncer.

La guerre informatique ne s'arrête pas là. Quelques très rares informaticiens ont acquis suffisamment de maîtrise pour *casser* les *protections*

(4) Les mémentos et agendas personnels, s'ils sont informatisés, devraient donc être déclarés à la CNIL.
(5) Les fichiers médicaux ne sont accessibles que par l'intermédiaire d'un médecin.
(6) On a dit parfois que la rédaction d'un livre de haut niveau, scientifique, historique, encyclopédique ... demandait à peu près autant d'années de travail que l'écriture d'un logiciel, vendu pourtant beaucoup plus cher à une clientèle beaucoup plus vaste.
(7) Ceux-ci savent très bien retrouver sur un disque des fichiers effacés ou détruits (du moins tant que la place libérée n'a pas été réutilisée pour d'autres fichiers).

logicielles (déplombage). Ils sont même capables de s'infiltrer sur presque n'importe quel réseau, malgré les mots de passe, et d'y créer les plus grands dommages. Toutefois, ces *hackers* (on pourrait les appeler des *casseurs*) semblent obéir à un code d'honneur et ont rarement exécuté leurs menaces, sinon pour montrer de quoi ils étaient capables. Leur comportement relève plus du jeu que de la malveillance. Néanmoins, en France du moins, un tel jeu serait violemment réprimé (par la loi Godfrain).

Plus répandu est le danger des **virus** informatiques. Ce mot désigne une forme de destruction du logiciel (rarement du matériel) par une minuscule séquence d'instructions *(microprogramme)* implantée à l'insu de l'utilisateur dans l'un de ses logiciels. Les motivations des auteurs de ces incrustations parasites ne sont pas claires, peut-être parce qu'elles varient selon le virus propagé : jeu, malveillance, vengeance, lutte sournoise contre les *pirates*, sont les mobiles les plus souvent avancés.

Quoi qu'il en soit, ce microprogramme est toujours doté de l'étonnante faculté de se *reproduire*, c'est-à-dire de provoquer sa propre recopie un peu n'importe où dans les mémoires de masse de son hôte.

Ces copies peuvent être déclenchées par différents facteurs, tels que la date, l'heure ou le nombre de copies du logiciel hôte. Toujours est-il qu'un jour ou l'autre le calculateur tombe en panne. On s'aperçoit alors que beaucoup de programmes sont en partie détruits ou inutilisables, ainsi que bien souvent leurs sauvegardes. On ne connaît pas beaucoup de parades à ce fléau : on peut installer dans son disque dur un *détecteur de virus* plus ou moins efficace, qui vérifie à chaque initialisation du système son intégrité, ou qui recherche la *signature* des virus connus. Une autre précaution consiste à n'introduire sur son disque dur ou sur ses disquettes que des logiciels dont on est sûr. Telles sont les recommandations des spécialistes. Mais comment être sûr d'un logiciel ?

Il est arrivé que des logiciels achetés tout à fait légalement soient infectés de virus par les soins d'un employé de l'éditeur voulant se venger d'un affront ou d'un licenciement. Dans le même ordre d'idées, on peut être victime d'un parasite opérant brusquement à retardement, une **bombe** en quelque sorte, ou un *cheval de Troie (Trojan horse)*. On dit que de tels pièges auraient été installés dans de gros logiciels écrits par des firmes pour des clients étrangers peu sûrs. Tant que le contrat est respecté, le service d'entretien retarde périodiquement l'action de la bombe. Le partenaire veut-il jouer cavalier seul ? A court terme le logiciel s'autodétruira.

La liste des désagréments associés à l'informatique n'est sûrement pas complète, tant sont grandes l'imagination et l'adresse de certains informaticiens. Beaucoup d'organismes se préoccupent de la protection de leur logiciel aussi bien contre les agressions externes que contre les malveillances possibles de la part de leurs employés. Une nouvelle profession est née, celle de *chargé de sécurité informatique*.

Les ennuis répertoriés ci-dessus constituent des cas extrêmes, concernant assez peu l'utilisateur modeste et scrupuleux. Il n'est cependant pas à l'abri de déboires plus classiques, tels que l'achat d'un logiciel, qui, malgré son prix élevé, ne tient pas ses *promesses* ou ne correspond pas à l'attente placée en lui. Il est bien rare que le producteur accepte de reprendre ce logiciel, ou même de l'échanger (bien qu'il s'en proclame toujours propriétaire). Si tant est qu'il sache le faire, le client n'aura même pas de droit de le corriger. Il pourra toujours signaler le défaut à l'éditeur, qui en tiendra peut-être compte pour mettre au point la prochaine version. Plusieurs mois après, il pourra acheter le complément qui modifiera le logiciel défaillant. Ce complément coûtera jusqu'à un ou deux tiers du prix initial, toujours sans aucune réelle *garantie*.

Contre ces mésaventures, peu de secours existent. Les lois ne prévoient pas grand-chose. Il ne faut acheter qu'avec prudence, sans trop se fier aux prospectus des vendeurs. Où se renseigner ? Un bon procédé consiste à lire la presse informatique, c.-à-d. les **revues spécialisés**. Mais on s'apercevra vite que, dans la plupart, les études de produits sont rarement critiques et que la "publicité" y a la part belle. De plus, il est bien rare qu'on y trouve une étude sur un logiciel en libre service *(shareware*, cf. §3) ou tombé dans le domaine public (c.-à-d. sans redevance, *freeware)*.

Il faut donc s'en remettre à l'indice de satisfaction de ses amis ou des autres usagers, en effectuant une moyenne pondérée. Une meilleure solution consisterait pour les consommateurs d'informatique à se regrouper en associations financièrement indépendantes des fournisseurs et capables d'effectuer des bancs d'essai et de les publier (comme c'est le cas pour la consommation ordinaire). Les grands organismes industriels ou publics ont œuvré un peu en ce sens, en faisant contrôler par un département *ad hoc* tous leurs achats informatiques. Cependant, le rôle de ces départements se borne le plus souvent à négocier, au nom de l'organisme, des contrats d'achats et des remises pour les employés. C'est ainsi que les gros clients (*les grands comptes*) obtiennent des remises de 40 ou même 50% sur les prix publics **(8)**, c'est-à-dire sur ceux appliqués aux particuliers ou aux PME.

Un grand organisme scientifique public (le CNRS) produit *le Micro-Bulletin*, revue consacrée à l'informatique vue par les clients. Elle a l'avantage d'être impartiale, les comptes-rendus de son laboratoire d'essai sont sans complaisance et le style très différent de celui des revues commerciales. Cependant, très orientée Unix, la revue pourra paraître à plus d'un amateur d'un niveau trop élevé.

(8) Ce qui les ramène souvent au prix pratiqué aux USA.

La loi de 1985 soulève une autre difficulté, celle devant laquelle sont placés les enseignants d'informatique. A moins de faire travailler leurs élèves les uns après les autres, il leur faut acheter un exemplaire de *chaque* logiciel par poste de travail, même si l'utilisation en est peu fréquente et sans bénéfice commercial. Même en tenant compte des *remises*, la législation actuelle ne peut que contribuer à freiner l'apprentissage de l'informatique en le rendant trop onéreux.

Nous espérons bien que certains lecteurs seront tentés par l'écriture de leurs propres logiciels. Peut-être voudront-ils les vendre ? Si le logiciel occupe un créneau libre (c.-à-d. traite un problème pour lequel il n'existe encore aucun logiciel), l'initiative peut être couronnée de succès. Mais si le nouveau-venu marche sur les brisées des grands, attention aux **procès pour plagiat** ! Le langage informatique est si peu nuancé que, pour parvenir à un même résultat, on utilise presque inévitablement les mêmes séquences. L'accusation de plagiat est donc facile. Ce genre de procès fait rage aux USA. Presque toujours d'ailleurs, les parties finissent par admettre qu'un *mauvais arrangement vaut mieux qu'un bon procès*, au grand dam de la concurrence.

Le milieu du logiciel est un monde impitoyable !

11 - LES LENDEMAINS QUI CHANTENT

Parvenus au terme de ce manuel, nous aimerions abandonner la simple relation des faits pour laisser entrevoir au lecteur ce que l'informatique pourrait modifier tant dans le pays que sur la planète. Cette dernière section sera donc consacrée à ce que d'aucuns appelleront la *prospective* et d'autres le rêve, selon leur degré d'optimisme. Bien sûr, les idées exposées ci-après ne reflètent que les opinions de l'auteur.

L'enjeu engagé par l'informatique est de taille. On a déjà compris dans la section 7 tout le bénéfice que nos **sociétés techniques** retireraient de la généralisation des réseaux et de l'abaissement de leur coût d'exploitation : meilleure utilisation de l'espace et du temps, économie des matières premières, décroissance des pollutions [(9)] ... Ce seront les idées qui voyageraient le long des câbles à la vitesse de la lumière et non plus leurs créateurs sur les routes à une vitesse d'escargot.

Pour les pays en voie de développement (**PVD**), il faut hélas regarder à plus long terme, mais il n'est pas exclu que l'informatique accélère leur transition vers le mieux-être occidental. Un début de preuve en est apporté par ces pays d'Extrême-Orient (et pas seulement le Japon) qui ont su saisir la balle au bond et jouer un rôle notoire dans le secteur du **matériel**. A ce propos, rappelons que le **logiciel** semble encore l'apanage de l'Occident. Cette situation ne saurait durer. La programmation exige, outre une compétence qui finira bien par diffuser, rationalisme, persévérance, amour du travail bien fait. Ce ne sont pas des qualités réservées aux riverains de l'Atlantique. D'autres partenaires pourraient bientôt entrer en scène. Déjà on peut recenser d'importantes sociétés de logiciel travaillant en Inde pour des firmes US. Il en va de même en Israël, où l'essor de l'industrie logicielle semble être lié à l'immigration d'informaticiens russes après l'éclatement du bloc de l'Est. Qui d'autre bientôt ?

Avant d'être les acteurs du développement informatique, les PVD en seront simplement **utilisateurs**. Ce sont les **réseaux** qui le leur permettront, surtout via les satellites. Grâce à eux, seront distribués sur toute la planète le savoir et le savoir-faire et pourra parvenir à ses commanditaires le produit de tout travail intellectuel. Ainsi, les réseaux permettraient de fixer sur leurs sites ancestraux des populations paysannes attirées par les mirages des mégapoles et créatrices de bidonvilles. Les réseaux rendraient viable un type d'économie mixte, à la fois agricole et technique, qui leur permettrait d'émerger de la misère. L'informatique pourrait leur éviter de passer, dans leur développement, par la phase de concentration industrielle qu'a connue l'Europe au XIXe siècle.

L'apport le plus direct sera celui des systèmes-experts. Au cœur de la brousse, ils répareront et entretiendront bientôt, sous contrôle humain, les machines complexes importées à grand frais, rentabilisant ainsi ces équipements. Bien sûr, il faudra permettre à tous les peuples l'acquisition du matériel nécessaire, récepteurs de satellites compris. Mais la baisse du prix de ce type de matériel au cours des dix dernières années est suffisamment encourageante pour que – le progrès technique aidant – l'accès à l'informatique de tout groupe social devienne d'un prix abordable [(10)].

Reste le problème du **logiciel**. Il est certain que, si l'on continue sur la lancée des années 80 en encourageant les prix élevés, le développement de l'informatique sera freiné, y compris dans nos pays. Le particulier, tout comme la PME, n'y auront qu'un accès limité et au prix d'un effort financier pas toujours envisageable. La prohibition actuelle concernant l'utilisation collective des logiciels est particulièrement mal venue. Une telle politique interdira la diffusion généralisée de l'informatique et spécialement vers le tiers monde.

(9) Par la réparation, qui ne peut devenir rentable qu'avec le dégrèvement du travail et la taxation des nuisances.

(10) On peut remarquer qu'entre 1990 et 95, le prix d'un ordinateur d'un type donné a baissé d'un facteur compris entre 5 et 10, celui d'une imprimante d'un facteur 4 à 6 (hormis les imprimantes PostScript, sans doute à cause des redevances). Pendant ce même laps de temps, le prix des logiciels a baissé de moins d'un facteur 2.

Un obstacle supplémentaire proviendra de la **langue**. Une **traduction** parfaite du savoir – et encore plus, sa diffusion – dans tous les dialectes du monde coûterait très cher. Bien sûr, on sait que les ordinateurs sont capables de traduction. Mais le résultat en est bien décevant, les contresens n'étant même pas exclus, tant les langues humaines font preuve de nuances allant jusqu'à l'irrationalisme. Or l'ordinateur est et restera avant tout **rationnel**. Il lui sera très difficile pendant longtemps encore d'effectuer des traductions sûres.

Si on veut donner à l'informatique toute sa puissance au plan mondial, il lui faudra un **langage** propre, rationnel, **universel**. Ce langage pourrait également servir aux humains : il deviendrait leur *seconde langue*, utilisable pour les échanges techniques, commerciaux ou diplomatiques. Les informaticiens qui ont suivi l'évolution des langages de haut niveau ont pu discerner chez eux deux tendances : celle à se rapprocher des langues humaines et celle à fusionner en une langue commune. On a pu déjà s'en rendre compte dans les chapitres IX et X. Or il existe déjà une langue qui satisferait toutes les exigences de l'informatique. Il faut *espérer* – c'est bien le mot qui convient ! – que ceux qui s'opposent à son adoption accepteront enfin de faire le pas nécessaire vers une compréhension égalitaire et universelle.

L'adoption d'un langage commun entraînera des économies substantielles en frais de traduction pour les instances supranationales. L'argent ainsi épargné pourra servir à financer un plan de **développement des logiciels** généraux ou d'ordre culturel, afin de supprimer les anomalies relatées dans la section précédente. Une telle tâche pourrait par exemple être confiée à L'UNESCO, les logiciels à finalité commerciale ou industrielle continuant bien sûr à subir la loi du marché.

Nombre d'informaticiens – et pas des moindres – se sont inquiétés de la mercantilisation excessive de l'informatique. Certes, il est normal que cette industrie produise du bien-être et même de la richesse. Mais de là à ce que, comme disent les médias, *un certain boss soit assis sur une montagne d'or* **(11)**, il y a de la marge ! Un puissant mouvement intellectuel se développe contre cette tendance. On a vu que la diffusion d'Internet, plus idéaliste que commerciale, procède d'une telle éthique, ainsi que l'implantation sur ce réseau mondial de **serveurs publics** *ftp anonymous*, *gopher*, WWW, ... On retrouve le même esprit dans le monde Unix où règnent entraide et cessions gratuites de logiciels (système d'exploitation Linux, traitement de texte T_EX par exemple et bien d'autres). La **distribution GNU** a même été créée pour diffuser des logiciels gratuits. Citons aussi le geste de la société Aladdin Entreprises qui a mis à la disposition de tous les remarquables outils *GhostScript* et GhostView (cf. chap. 14). Aux Etats-Unis, ces humanistes se regroupent dans la *Free Software Foundation* ou dans la *League for Programming Freedom*, associations prônant la gratuité du logiciel ou tout au moins son coût modéré. Ces idées devraient faire leur chemin...

Revenant à l'échelon plus modeste de l'individu, nous voudrions terminer ce livre comme un conte de fées en formulant trois vœux à l'intention des usagers de l'informatique et surtout des plus jeunes. En premier lieu, nous souhaitons que la fréquentation de ces machines, ô combien rigides et strictes, communique aux humains un peu de leur **rationalisme**, porteur de rigueur intellectuelle, sans laquelle il n'y a pas de justice.

Le deuxième vœu est complémentaire du premier. Ces nouvelles machines vont, comme on l'a déjà dit, se charger de beaucoup de tâches absorbantes, fastidieuses, jusque-là confiées aux humains. Du temps de loisir sera ainsi dégagé et l'esprit libéré pour d'autres occupations. Nous souhaitons que ces **libérations** se fassent au bénéfice des productions les plus nobles du cerveau, la réflexion, la création, la poésie ... et que le *chiffre* cède le pas devant la *lettre*. On trouvera peut-être une contradiction entre ces deux vœux. Ils ne font pourtant que refléter deux des facettes du génie humain, *l'esprit de finesse* et *l'esprit de géométrie*. Ce n'est pas Blaise Pascal qui nous contredirait, lui qui décidément nous aura accompagnés tout au long du chemin.

Pour finir, souhaitons que les législateurs nationaux et internationaux sachent mettre en application les idées contenues dans le préambule de la loi *informatique et libertés* **(12)** :

L'informatique doit être au service de chaque citoyen.
Son développement doit s'opérer dans le cadre de la coopération internationale.

(11) Et devienne en quelques années, paraît-il, l'homme le plus riche du monde !
(12) *Journal Officiel de la République Française* (7 janvier 1978, page 227)

ANNEXES

CARACTERES ASCII

hex	*0..*	*1..*	*2..*	*3..*	*4..*	*5..*	*6..*	*7..*
..0	*Nul* 0	*Dle* ▶ 16	32	0 48	@ 64	P 80	‘ 96	p 112
..1	*Soh* ☺ 1	*Xon* ◀ 17	! 33	1 49	A 65	Q 81	a 97	q 113
..2	*Stx* ☻ 2	*DC2* ↕ 18	" 34	2 50	B 66	R 82	b 98	r 114
..3	*Etx* ♥ 3	*Xof* ‼ 19	# 35	3 51	C 67	S 83	c 99	s 115
..4	*Eot* ♦ 4	*DC4* ¶ 20	$ 36	4 52	D 68	T 84	d 100	t 116
..5	*Enq* ♣ 5	*Nak* § 21	% 37	5 53	E 69	U 85	e 101	u 117
..6	*Ack* ♠ 6	*Syn* ▬ 22	& 38	6 54	F 70	V 86	f 102	v 118
..7	*Bel* ● 7	*Etb* ↨ 23	’ 39	7 55	G 71	W 87	g 103	w 119
..8	*BS* ◘ 8	*Can* ↑ 24	(40	8 56	H 72	X 88	h 104	x 120
..9	*HT* ○ 9	*EM* ↓ 25	) 41	9 57	I 73	Y 89	i 105	y 121
..A	*LF* ◙ 10	*Sub* → 26	* 42	: 58	J 74	Z 90	j 106	z 122
..B	*VT* ♂ 11	*Esc* ← 27	+ 43	; 59	K 75	[91	k 107	{ 123
..C	*FF* ♀ 12	*FS* ∟ 28	, 44	< 60	L 76	\ 92	l 108	| 124
..D	*RC* ♪ 13	*GS* ↔ 29	- 45	= 61	M 77	] 93	m 109	} 125
..E	*SO* ♫ 14	*RS* ▲ 30	. 46	> 62	N 78	^ 94	n 110	~ 126
..F	*SI* ☼ 15	*US* ▼ 31	/ 47	? 63	O 79	_ 95	o 111	127

Le numéro décimal du caractère est inscrit directement dans sa case tandis que son numéro hexadécimal s'obtient en accolant le numéro de la colonne à celui de la ligne.

EXTENSION IBM

hex	*8..*	*9..*	*A..*	*B..*	*C..*	*D..*	*E..*	*F..*
..0	Ç 128	É 144	á 160	▒ 176	└ 192	╨ 208	α 224	≡ 240
..1	ü 129	æ 145	í 161	▒ 177	┴ 193	╤ 209	β 225	± 241
..2	é 130	Æ 146	ó 162	▓ 178	┬ 194	╥ 210	Γ 226	≥ 242
..3	â 131	ô 147	ú 163	│ 179	├ 195	╙ 211	π 227	≤ 243
..4	ä 132	ö 148	ñ 164	┤ 180	─ 196	╘ 212	Σ 228	⌠ 244
..5	à 133	ò 149	Ñ 165	╡ 181	┼ 197	╒ 213	σ 229	⌡ 245
..6	å 134	û 150	ª 166	╢ 182	╞ 198	╓ 214	µ 230	÷ 246
..7	ç 135	ù 151	º 167	╖ 183	╟ 199	╫ 215	γ 231	≈ 247
..8	ê 136	ÿ 152	¿ 168	╕ 184	╚ 200	╪ 216	Φ 232	° 248
..9	ë 137	Ö 153	⌐ 169	╣ 185	╔ 201	┘ 217	Θ 233	● 249
..A	è 138	Ü 154	¬ 170	║ 186	╩ 202	┌ 218	Ω 234	• 250
..B	ï 139	¢ 155	½ 171	╗ 187	╦ 203	▉ 219	δ 235	√ 251
..C	î 140	£ 156	¼ 172	╝ 188	╠ 204	▄ 220	∞ 236	n 252
..D	ì 141	¥ 157	¡ 173	╜ 189	═ 205	▌ 221	∅ 237	2 253
..E	Ä 142	Pt 158	« 174	╛ 190	╬ 206	▐ 222	ε 238	■ 254
..F	Å 143	*f* 159	» 175	┐ 191	╧ 207	▀ 223	∩ 239	255

EXTENSION MACINTOSH

hex	*8..*	*9..*	*A..*	*B..*	*C..*	*D..*	*E..*	*F..*
..0	Ä 128	ê 144	† 160	∞ 176	¿ 192	– 208	‡ 224	240
..1	Å 129	ë 145	° 161	± 177	¡ 193	— 209	· 225	Ò 241
..2	Ç 130	í 146	¢ 162	≤ 178	¬ 194	“ 210	, 226	Ú 242
..3	É 131	ì 147	£ 163	≥ 179	√ 195	” 211	„ 227	Û 243
..4	Ñ 132	î 148	§ 164	¥ 180	*f* 196	‘ 212	‰ 228	Ù 244
..5	Ö 133	ï 149	• 165	µ 181	≈ 197	’ 213	Â 229	ı 245
..6	Ü 134	ñ 150	¶ 166	δ 182	Δ 198	◌̈ 214	Ê 230	ˆ 246
..7	á 135	ó 151	ß 167	Σ 183	» 199	◌̸ 215	Á 231	˜ 247
..8	à 136	ò 152	® 168	Π 184	« 200	ÿ 216	Ë 232	¯ 248
..9	â 137	ô 153	© 169	π 185	… 201	Ÿ 217	È 233	˘ 249
..A	ä 138	ö 154	™ 170	∫ 186	202	Ú 218	Í 234	˙ 250
..B	ã 139	õ 155	´ 171	ª 187	À 203	¤ 219	Î 235	° 251
..C	å 140	ú 156	¨ 172	º 188	Ã 204	‹ 220	Ï 236	¸ 252
..D	ç 141	ù 157	≠ 173	Ω 189	Õ 205	› 221	Ì 237	˝ 253
..E	é 142	û 158	Æ 174	æ 190	Œ 206	fi 222	Ó 238	˛ 254
..F	è 143	ü 159	Ø 175	ø 191	œ 207	fl 223	Ô 239	ˇ 255

Les 128 premiers caractères sont ceux de la table ASCII.

Annexe 2

PERIPHERIQUE EN LIAISON SERIE

Le problème de la connexion par liaison série **type RS232** entre un calculateur et un périphérique a fait trébucher suffisamment de débutants pour qu'on lui consacre une annexe. Cette question est importante, puisque cette liaison permet de relier un ordinateur à un très grand nombre d'appareils. Presque toujours, le périphérique est beaucoup plus lent que le calculateur et des problèmes de synchronisme se posent. En premier lieu, il faudra s'assurer que les fils 2 et 3 du câble sont bien *croisés* entre eux si les connecteurs sont de même type, et non croisés dans le cas contraire.

Le fabricant du périphérique a prévu au moins **un moyen** pour assurer un fonctionnement correct. Les ordres arrivant à l'appareil souvent bien plus vite qu'il ne peut les exécuter, ils s'accumulent dans sa *mémoire tampon*. Lorsque le taux de remplissage de celle-ci atteint un certain seuil, le périphérique désactive une ligne (il la porte au potentiel du 0 logique) et la réactive lorsque son remplissage descend sous un autre seuil. Cette ligne s'appelle **DTR** *(data terminal ready)*. Ces deux seuils sont parfois programmables ; par défaut, ils sont souvent fixés à 80 et à 50% de la capacité-mémoire, ce qui implique qu'en cas de dépassement du seuil haut, on peut encore envoyer un volume d'instructions égal à sa **capacité résiduelle**, c.-à-d. par défaut à 20% de la capacité totale.

De plus, on peut en général demander au périphérique, par programmation, d'envoyer alternativement les caractères *Xoff* et *Xon* sur sa *ligne normale de sortie* (borne 3) au moment où ces seuils sont franchis. Ces deux moyens permettent une **concertation** entre appareils car, si l'ordinateur continuait d'émettre vers un phériphérique lent dont la mémoire est pleine, l'information transmise serait en partie perdue.

1 - Pauses

On peut ignorer les moyens ci-dessus ou ne pas savoir s'en servir. On essaiera alors d'introduire des **points d'arrêt** dans le programme, si (et seulement si) on en a la *source*. L'émission d'ordres s'arrêtera et le périphérique aura le temps de vider sa mémoire tampon. Quand il se sera lui aussi arrêté, on relancera le programme. Les points d'arrêt s'introduisent en Basic par l'ordre **STOP**, en Pascal par un **READKEY** et en C par **getch()**. On reprend l'exécution du Basic en frappant **CONT** au clavier (ou la touche F5 si l'on est en mode éditeur). Dans les autres langages, on frappe une touche quelconque.

2 - Utilisation de Xon-Xoff

On suppose qu'on est en *mode fichier* (cf. chap XIII, §3,) et en Basic. Après l'instruction **OPEN COM**, on enverra l'ordre prévu pour *activer* le mode Xon-Xoff, par exemple en HPGL

PRINT #*nf***, CHR$(27), ".N;19:", CHR$(27), ".I80;;17:"**

Le périphérique enverra le caractère *Xoff* (ASCII 19) quand le seuil d'alerte de sa mémoire tampon sera franchi. Mais, sur sa ligne normale d'émission, il peut envoyer également d'autres informations. Il faudra donc trier les caractères *Xon-Xoff* dans le flot des signaux parvenant au PC.

On enverra au périphérique les ordres par paquets assez volumineux (chaînes d'au moins 20 caractères, mais d'une longueur au plus égale à la taille de la mémoire tampon résiduelle). Avant chaque émission, on s'astreindra aux vérifications suivantes :

– lire la valeur du nombre d'octets contenus dans le fichier de réception *#nf* avec l'ordre *m* **= LOC (***nf***)**

– si *m* > 0, lire le contenu de *#nf* avec *ch* **= INPUT$ (***m* **, #***nf***)**

– rechercher (grâce à la fonction **MID$**) si la chaîne *ch* contient le caractère *Xoff* (**CHR$(19)**) ET ne contient pas ensuite *Xon* (**CHR$(17)**). Si c'est le cas, recommencer cette procédure jusqu'à ce qu'un caractère *Xon* ait été reçu. N'écrire dans le fichier *#nf* (émission vers le périphérique) que si, de *Xon* et de *Xoff*, *Xoff* n'est pas le dernier caractère reçu.

Ce procédé plutôt laborieux ne marche pas toujours très bien, à cause des nombreux codes pouvant être envoyés par le périphérique mêlés aux caractères *Xon-Xoff*. Les procédés suivants sont plus simples et plus efficaces.

3 - Utilisation du DTR en Basic

Comme on l'a exposé dans le chapitre XV, trois fils électriques suffisent à assurer une liaison série ordinaire. Mais des lignes supplémentaires seront fort utiles. Ainsi, dans le cas d'un périphérique lent, il est avantageux de réunir par une 4e ligne la sortie DTR du périphérique à la borne 6 (DSR) ou à la borne CTS (n°5 ou n°8 respectivement selon que le connecteur a 25 ou 9 broches) – ou aux deux – du connecteur série utilisé sur le calculateur.

En *mode fichier*, l'*interpréteur* du Basic n'émet vers le périphérique que si les bornes précitées sont à la bonne tension. Ce procédé serait parfait si le Basic ne cessait pas ses envois dès le basculement desdites lignes, même au beau milieu d'un mnémonique. Certains phériphériques n'acceptent pas une telle troncature, car ils vérifient la validité des instructions dès leur arrivée ; détectant des codes erronés, ils signalent une erreur et *se plantent*. Avec des appareils aussi vicieux, il faut utiliser d'autres moyens.

4 - Utilisation programmée du DTR

Avant l'envoi de **chaque chaîne** (et non avant chaque caractère), on examinera la valeur du 5e ou du 6e bit du 6e registre de la carte d'entrée-sortie. Ces bits reflètent l'état des bornes précitées. On attend pour envoyer la chaîne que ce ou ces bits soient à 1 (la chaîne ne devra pas dépasser la taille résiduelle de la mémoire). On risque d'attendre indéfiniment *(plantage)* si le périphérique est en défaut. Aussi, vaut-il mieux, avant tout envoi d'ordres, passer d'abord par un sous-programme qui attend que le DTR soit à 1, mais pas plus d'une minute par exemple. Après avoir inhibé le fonctionnement automatique en plaçant les options **CS,RS,CD,DS** à la fin de l'**OPEN COM** (cf. chap. XIII, §3a, p. 119), on écrira :

```
.....   GOSUB 1000:  PRINT #nf, chaîne
.....
1000    ON TIMER(60)  GOSUB 2000 :  TIMER ON
1001    lect = INP (adr+6)
1002    IF lect AND &H10 = 0    THEN  GOTO 1001
1003    TIMER OFF :   RETURN
.....
2000    PRINT "Traceur hors service." : STOP : RETURN
```

L'unique inconvénient de ce procédé est *son manque de portabilité* puisqu'il utilise une adresse-machine. Sur un PC, cette adresse *adr* est **&H3F8** ou **&H2F8** selon qu'on est connecté à COM1 ou à COM2.

5 - Procédé Xon-Xoff avec interruptions

C'est de loin la meilleure façon de gérer la communication sous *Xon-Xoff*, si l'emploi du DTR n'est pas possible. On écrit un petit programme qui, sur un PC, à chaque interruption n°4 (avec COM1, n°3 avec COM2), ira vérifier (premier bit de *adr*+5) qu'un caractère a bien été reçu. Si oui, il le lira (dans *adr*) et l'examinera. S'il s'agit de *Xoff*, il rendra fausse une variable d'autorisation (*aut*); si c'est *Xon*, il la mettra à vrai. Ce programme remplacera le vecteur d'interruption correspondant du BIOS, puis on démasquera le PIC. Avant tout envoi de chaîne, on vérifiera la valeur de *aut* : si elle est fausse, il faudra attendre qu'elle devienne vraie, mais toujours durant un temps limité. La description complète du procédé sortirait du cadre de cet ouvrage.

6 - Programmation détaillée des registres de l'ERAU

Voici le détail d'un programme de communication série en protocole câblé. L'adresse d'implantation de la carte est *adr* (3F8, 2F8, 3E8, $2E8_{16}$ pour les sorties COM1, 2, 3 et 4 sur PC). On trouve deux types de connecteurs: DB25 avec 25 broches, DB9 avec 9. Assurer les 4 liaisons indispensables :

1 – Relier les masses (bornes 7 si DB25, 5 si DB9).
2 – Si connecteurs différents, relier 2-2, 3-3,
sinon 2-3 et 3-2.
3 – Relier | la borne CTS du PC (5 si DB25, 8 si DB9)
au DTR du périph. (20 si DB25, 4-DB9).

On utilisera le Basic avec ses ordres d'ES directs :

OUT *adr*, *oct*	*oct* = **INP (***adr***)**	équivalents à
outportb (*adr*,*oct***);**	*oct* = **inportb (***adr***);**	en C,
PORT [*adr***] :=** *oct*;	*oct* **:= PORT [***adr***];**	en Pascal.

Initialisation :

```
OUT adr+3, &H80    ' met à 1 le bit 7 reg. 3 (DLAB).
OUT adr, 115200/deb   ' initialise débit, si deb>450 bd
OUT adr+3, 4*(bar + par*(par+1)) + bdon - 9
```
remet DLAB à 0 , initialise la parité *(par* = 2, 1, 0 → paire, impaire, nulle), le nombre de bits de données *(bdon* = 8 ou 7) et celui d'arrêt *(bar* = 2 ou 1).

Lecture du caractère *car* **:**

```
c = INP (adr+5) :       t = 0
WHILE (c AND 1) = 0      ' aucun octet n'est en attente
   t = t+1:  retard:  c = INP(adr+5):  si (t>tmax)   arrêt
WEND                     ' un caractère a été reçu
car = INP (adr)          ' on le lit
```

Envoi de la chaîne *ch*$ **de** *n* **caractères :**

```
c  = INP (adr+6) :  t = 0 :  f = &H30
                    ' f filtrera les 2 bits utiles
WHILE ((c AND f ) <> f )  retard:
      c=INP(adr+6):   t = t+1:   si  (t>tmax)   arrêt
WEND  ' CTS=1, le tampon récepteur n'est plus saturé

FOR i = 1 TO n
    c = INP (adr+5) :          t = 0
    WHILE   c AND &H60 <> &H60
          retard:   c = INP(adr+5):
          t = t+1:  si (t>tmax)    arrêt
    WEND      ' les 2 registres d'émission sont vides
    OUT adr, ASC(MID$ (ch$,i,1))
                    '.ch[i] en Pascal & C
NEXT i ' n inférieur à la capacité résiduelle du tampon
```

NB : Les retards, fonctions du débit, sont de quelques millisecondes. Au lieu de générer un retard, on peut mesurer le temps écoulé, comme dans l'exemple ci-dessus section 4 (on introduit une valeur numérique dans la chaîne *ch*$ avec **STR$ (***var***)** .

Annexe 3

CODE A BARRES

Les lecteurs intrigués par le code-barre trouveront dans cette annexe quelques détails relatifs à cette codification mentionnée page 142, chapitre XVI.

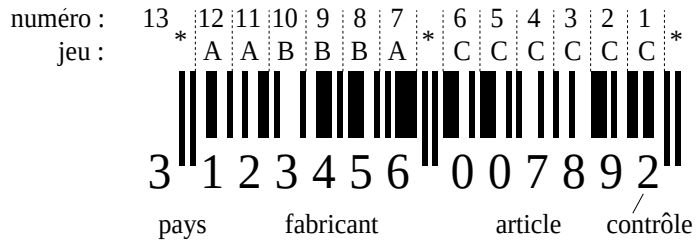
Le code-barre est un **graphisme** identificateur imprimé ou collé sur l'emballage d'un produit vendu au détail. Il doit pouvoir être lu sans erreur par un lecteur optique (crayon laser par exemple), bien que la vitesse et la direction de son déplacement puissent varier dans de larges limites. Divers codes-barres sont en service. Le plus répandu est l'**EAN13**, normalisé en Europe (EAN) et en Amérique du Nord (UPC).

L'EAN-13 représente **13 chiffres**, qu'on numérotera ici de 1 à 13 à partir de la droite. La signification de ces chiffres est la suivante :

- le 13e et le 12e identifient le **pays** d'origine selon le tableau au verso,
- les chiffres suivants (en général de 6 à 11 compris) forment un numéro identifiant dans ce pays le **fabricant** de l'article,
- les chiffres 2...5 forment le numéro donné au **produit** par ce fabricant,
- le premier chiffre est un **caractère de contrôle** qui dépend de la valeur des chiffres précédents. Il est calculé pour que la somme des chiffres de rang impair (13, 11, ... 1), plus 3 fois la somme des chiffres de rang pair (12, 10, ... 4, 2) soit un multiple de 10.

00 01 03 04 06 09	USA, Canada	76	Suisse
30 à 37	France	80 à 83	Italie
40 à 43	Allemagne	84	Espagne
49	Japon	860-861	Yougoslavie
50	Grande-Bretagne	87	Pays-Bas
54	Belgique et Luxembourg	90-91	Autriche
57	Danemark	93	Australie
600	Afrique du Sud	94	N-Zélande
64	Finlande	978	Livres ISBN
70	Norvège	02	**(1)**
73	Suède	20 à 29	**(1)**

Le graphisme est inscrit dans un rectangle en principe de 31,35 sur 22,9 mm, mais il peut être "dilaté" avec un facteur *G* variable de 0,8 à 2. Il est formé de 30 barres sombres entrelacées avec 29 barres claires. La largeur de chaque barre est multiple d'une valeur appelée *module* (0,33 mm pour *G*=1). La largeur propre du graphisme est de 95 modules. Il est précédé par une marge claire de 11 modules et suivi par une autre de 7 modules, portant la largeur occupée à 113 modules.



Exemple de graphisme code-barre

Les 59 barres représentent 15 symboles : 12 chiffres et 3 séparateurs. Les **séparateurs**, notés * dans la figure précédente, sont situés en début, fin et milieu de dessin. Ils dépassent souvent les autres par le bas et délimitent deux **champs**, droite et gauche.

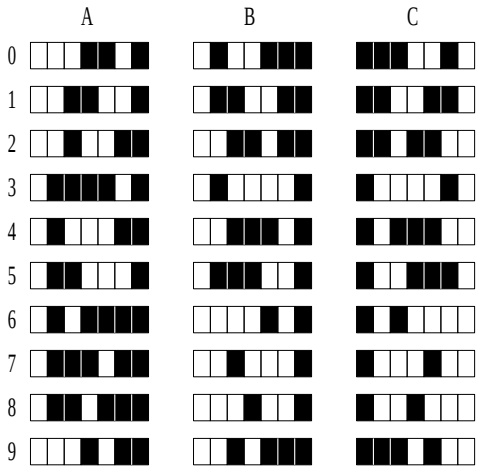
Les séparateurs extrêmes sont formés chacun de deux barres sombres encadrant une barre claire et le séparateur central de deux barres sombres entre trois barres claires. Chacune de ces barres a pour largeur un module (on peut dire *un bit*). La largeur des séparateurs est donc de 5 ou 3 bits.

(1) Le code 02 est réservé aux marchandises vendues au poids ; les codes 20-29 aux articles codifiés en magasin.

Chacun des champs contient 6 **chiffres décimaux**, tous formés de 7 bits et codés par 4 barres.

Dans le champ de droite, le code appliqué est le jeu C donné par la figure ci-contre. Ce code est appelé pair parce que le nombre de modules sombres est pair (2 ou 4 par chiffre). Il commence par une barre noire.

Dans le champ de gauche, on utilise les deux codes A et B. Le jeu A est un code impair et c'est cette qualité qui permet à l'informatique de savoir dans quel sens elle doit interpréter les signaux lus optiquement. Le code B est le symétrique du C et le A en quelque sorte son négatif.



Codage du 13e chiffre selon la répartition des jeux A et B.

La distribution des jeux A et B dans le champ de gauche n'est pas fantaisiste. C'est elle qui donne la valeur du **13e chiffre** (appelé *chiffre extérieur*, car il n'est pas codé par des barres) selon le tableau ci-contre.

13	**12**	**11**	**10**	**9**	**8**	**7**		**6**	**5**
0	A	A	A	A	A	A		C	C
1	A	A	B	A	B	B		C	C
2	A	A	B	B	A	B		C	C
3	**A**	**A**	**B**	**B**	**B**	**A**		**C**	**C**
4	A	B	A	A	B	B		C	C
5	A	B	B	A	A	B		C	C
6	A	B	B	B	A	A		C	C
7	A	B	A	B	A	B		C	C
8	A	B	A	B	B	A		C	C
9	A	B	B	A	B	A		C	C

Annexe 4

UN PEU DE MATHS ...

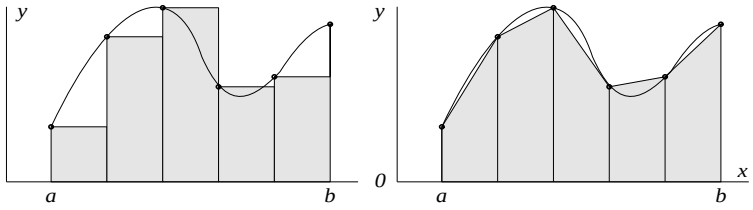
L'étudiant scientifique pourra trouver ici quelques notions sur des méthodes couramment employées en programmation. Elles relèvent de l'**analyse numérique**, branche des mathématiques très ancienne, mais développée surtout depuis l'avènement des machines à calculer. Ces méthodes, d'une utilité incontestable, mais parfois limitée, permettent de trouver, grâce aux seules quatre opérations fondamentales de l'arithmétique, des solutions approchées à des problèmes non résolus par l'algèbre.

Ces méthodes sont démontrées dans les nombreux traités d'analyse numérique, qui sont en général de lecture difficile. Pourtant, ces méthodes peuvent être utiles à bien des amateurs non formés aux mathématiques abstraites. C'est pourquoi nous en citerons quelques-unes ici, bien sûr sans les démontrer, mais en donnant suffisamment d'explications pour les faire comprendre de manière intuitive.

1 - Intégration numérique

Lorsqu'une fonction algébrique $f(x)$ définie sur l'intervalle $[a,b]$ n'est pas intégrable, on peut utiliser une méthode numérique calquée sur la définition de l'intégrale, limite de la somme de l'aire des rectangles de côtés $\Delta x = (b-a)/N$ et $y = f(a + i\Delta x)$ avec $0 < i \leq N$, quand $N \to \infty$ (N et i entiers) :

$$I(a,b) = \int_a^b f(x)\,dx = \lim \sum_{i=0}^{(b-a-\Delta x)/\Delta x} f(a+i\Delta x)\,\Delta x \qquad \text{quand } \Delta x \to 0$$



L'intégrale étant égale à l'aire comprise entre la courbe $f(x)$ et l'axe y=0, la méthode consiste à approcher cette aire par une somme de rectangles de largeur uniforme Δx tels que les montre la figure de gauche. Ce procédé, appelé **méthode des rectangles**, est peu employé, car il converge trop lentement (sous-entendu : vers le bon résultat). Plus précise est la **méthode des trapèzes**, très simple, selon laquelle

$$I(a,b) = (\tfrac{1}{2}\, y_{\mathbf{0}} + y_{\mathbf{1}} + y_{\mathbf{2}} + y_{\mathbf{3}} + \ldots + y_{\mathbf{n-1}} + \tfrac{1}{2}\, y_{\mathbf{n}})\,\Delta x$$

où Δx est petit et où $y_{\mathbf{i}}$ signifie $f(a+i\Delta x)$. On le comprendra à l'aide de la figure de droite, qui montre comment cette relation approche par des trapèzes la surface étudiée (pour retrouver la formule des rectangles, faire $\tfrac{1}{2}\, y_{\mathbf{0}} = y_{\mathbf{0}}$ et $\tfrac{1}{2}\, y_{\mathbf{n}} = 0$). D'autres procédés sont encore meilleurs, telle la **méthode de Simpson**, pas trop difficile à programmer, approchant $f(x)$ par des arcs de paraboles,

$$I = (y_{\mathbf{0}} + 4y_{\mathbf{1}} + 2y_{\mathbf{2}} + 4y_{\mathbf{3}} + \ldots + 2y_{\mathbf{N-2}} + 4y_{\mathbf{N-1}} + y_{\mathbf{N}})\,\Delta x/3$$

avec N pair. On progresse par étapes, en diminuant à chaque fois la valeur Δx (par exemple, en la divisant par 2), donc en augmentant le nombre de points : on arrête le calcul lorsque le résultat d'une étape ne diffère du précédent que d'une valeur inférieure à la précision recherchée.

2 - Valeur approchée d'une fonction

Si on connaît la valeur de $f(x)$ et celle de ses dérivées pour $x=x_0$, on peut calculer $f(x)$ au voisinage de x_0 par la formule de Taylor :

$$f(x) = f(x_0) + (x-x_0)\,f'(x_0) + \frac{(x-x_0)^2}{2!}f^{(2)}(x_0) + \frac{(x-x_0)^3}{3!}f^{(3)}(x_0) + ...$$

L'écart avec la valeur exacte est en général de l'ordre de grandeur du premier terme négligé, soit $(x-x_0)^n\, f^{(n)}(x_0)\,/n!$, facilement calculable. C'est ainsi que sont programmées les fonctions mathématiques des bibliothèques fournies avec les logiciels de programmation.

Voici, par exemple, la méthode employée pour le calcul de sinus x. On divise le cercle trigonométrique en 8 secteurs de 45° centrés sur les angles $x_0 = 0$, $\pi/4$, $\pi/2$, $3\pi/4$..., pour lesquelles on connaît la valeur du sinus et de ses dérivées successives. Si $x < \pi/4$, on utilise la relation (avec x en radians)

$$\sin x = x - x^3/3! + x^5/5! - x^7/7! + x^9/9! - ...$$

Le terme $x^9/9!$ est au plus égal à $5{,}8\ 10^{-10}$ et celui en $x^{13}/13!$ à $8{,}5\ 10^{-16}$. On peut donc se limiter ici à 4 termes en simple précision et à 6 ou 7 en double.

Si $x > \pi/4$, on place x dans le secteur correspondant et on utilise la formule de Taylor avec la variable $(x-x_0)$ qui ne dépassera jamais 22,5°, soit 0,39 rad. On est ramené au problème précédent avec d'autres valeurs pour les dérivées, mais dont la valeur absolue ne dépassera jamais 1. Ce procédé est très employé, mais n'est pas toujours aussi simple, car les dérivées successives $f^{(n)}$ du sinus sont égales à 0, 1 ou –1.

3 - Résolution d'une équation

On cherche une valeur de x satisfaisant l'équation

$$f(x) = a$$

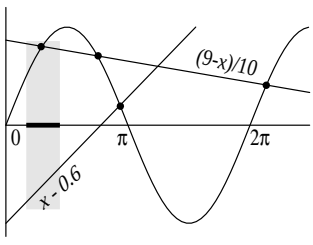
avec a nul ou constant (forme à laquelle se ramènent de nombreux problèmes). Cette relation n'est pas toujours résoluble par l'algèbre. Des procédés numériques sont alors employés, telle la méthode **par dichotomie**. On commence *au jugé* avec deux valeurs x_1 et x_2 encadrant la solution, i.e. telles que $f(x_1)<a$ et $f(x_2)>a$. Puis on calcule $x_3 = (x_1+x_2)/2$ et $f(x_3)$. Des 3 points, x_1, x_2 et x_3, on ne garde que les deux encadrant au plus près la solution et on recommence.

La dichotomie est moins rapide que la **méthode de Newton**, décrite page 93, mais cette dernière peut présenter de grosses difficultés quand la fonction n'est pas simple. En effet, les solutions peuvent être multiples et on convergera vers l'une ou l'autre selon le point de départ. On peut d'ailleurs ne pas converger du tout. Il est prudent de programmer un graphique d'accompagnement quand c'est possible.

Il faut se faire d'abord une idée précise du nombre de solutions et de leur localisation approximative en étudiant le comportement de $f(x)$ (points marquants, extremum...). A titre d'exemple, on pourra s'exercer sur la relation pourtant simple

$$\sin \pi x = ax + b$$

(fig. ci-contre) illustrant l'importance du point de départ.



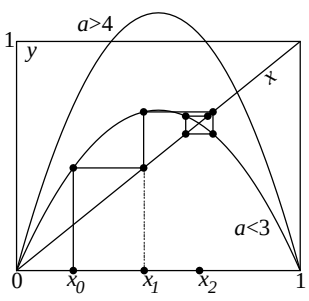
Si $a=1$ et $b=-0.6$ par ex., tout départ hors du domaine [0,525 ; 1,39] fait diverger de façon incohérente.

Il arrive pourtant qu'après beaucoup de pas (200, 500 ...), on rentre dans le bon intervalle et on obtienne vite la solution si elle est unique. Mais si elle est multiple, comme avec la droite $y = (9-x)/10$ ci-dessus, sans précautions, on tombe sur n'importe laquelle.

La méthode de Newton est itérative, mais elle exige le calcul d'une dérivée. Dans certains cas, le problème peut se mettre sous la forme particulière

$$f(x) = x$$

On peut alors itérer de façon plus simple, en donnant à x une première valeur x_0, puis en calculant $f(x_0)$, d'où une nouvelle valeur x_0, puis $f(x_0) \rightarrow x_0$, etc. La convergence est bien plus lente qu'avec la méthode de Newton. Elle est représentée dans la figure cidessous avec $f(x) = ax(1-x)$. Les points cherchés sont ceux où la droite $y = x$ coupe la parabole $y = f(x)$ (cet exemple est un peu artificiel, puisque, avec une telle fonction, contrairement à la relation précédente, la solution peut se calculer par l'algèbre).



Mais cette itération est exemplaire parce qu'elle se comporte de façon incohérente et chaotique, comme on s'en rendra facilement

compte par construction graphique, dès que $a>3$ (ceci est lié au fait que la dérivée en valeur absolue est supérieure à 1 au point solution). Cet exemple, qui devrait nous apprendre à rester méfiants envers les procédés d'itération, sert souvent d'introduction aux *théories du chaos*.

4 - Résolution d'un système d'équations linéaires

Soit un système d'équations linéaires tel que :

$$\begin{aligned} a_1\,x + b_1\,y + c_1\,z &= d_1 \\ a_2\,x + b_2\,y + c_2\,z &= d_2 \\ a_3\,x + b_3\,y + c_3\,z &= d_3 \end{aligned}$$

où les coefficients a_j , b_j , c_j (plus généralement les a_{ij}) et les valeurs d_j sont des constantes. Pour le résoudre, on peut calculer le déterminant du tableau (matrice) formé par les coefficients a_{ij} ainsi que ses mineurs. Il est bien plus rapide d'*inverser* la matrice (si elle existe) ou plus exactement de multiplier les deux membres du système par la matrice inverse a_{ij}^{-1},

$$x_i\,.\,a_{ij} = d_j \quad \rightarrow \quad x_i\,.\,\mathbf{1} = a_{ij}^{-1}.\,d_j$$

Il faut ramener la matrice a_{ij} à un tableau formé de zéros, sauf sur sa diagonale dont les a_{jj} seront égaux à 1 (matrice unité). On procède par itération de ligne à ligne en divisant tous les termes de la ligne j par le *pivot* a_{jj} (élément diagonal), puis en soustrayant la nouvelle ligne j de chacune des autres lignes k après multiplication par le terme a_{kj} , pour annuler les termes non-diagonaux de la colonne j.

Cette méthode est rigoureuse et se programme facilement, mais le grand nombre d'opérations impliquées (avec une précision finie) rend son résultat parfois décevant. La méthode du *pivot maximum* améliore nettement ce procédé : elle consiste, avant chaque étape, à permuter des lignes (ou même lignes et colonnes) pour que le nouveau pivot ait la plus grande valeur possible, ce qui améliorera la précision dans les divisions ultérieures, mais complique la programmation.

5 - Interpolation d'une fonction.

Souvent, une fonction n'est connue que par un certain nombre n de ses points y_i . On peut désirer lui trouver une expression analytique pour la traiter plus commodément *(ajustement, lissage*, en anglais *fit)* ou pour évaluer des points intermédiaires *(interpolation)*. Dans certains cas (fonction périodique, analyse harmonique, convolution...), il sera judicieux de la décomposer en série de Fourier, dont les coefficients se prêtent bien au calcul numérique **(1)**.

Parfois, on connaîtra la loi $f(x)$ à laquelle obéit le phénomène, mais seulement à des constantes près. On tentera alors de ramener le problème à la méthode des moindres carrés. Celle-ci permet d'obtenir un bon ajustement en exprimant que la somme $\mathbf{S} = \sum \varepsilon_i^2$ du carré des écarts $\varepsilon_i = f(x_i) - y_i$ entre la courbe lissée $f(x)$ et les points donnés y_i passe par un minimum. Pour cela, si f ne dépend que de quelques paramètres inconnus a, b..., on écrit que les dérivées partielles $\partial\mathbf{S}/\partial a$, $\partial\mathbf{S}/\partial b$ ... sont nulles, ce qui conduit à un système d'équations parfois résoluble.

Dans le cas simple où la fonction est linéaire, $f = ax+b$, $\mathbf{S} = \Sigma\,[ax_i+b-y_i]^2$, on dérive facilement $\mathbf{S}$ pour obtenir comme solution

$$\begin{aligned} a &= (nS_{xy} - S_xS_y) \,/\, (nS_{xx}-S_xS_x) \\ b &= (S_yS_{xx} - S_{xy}S_x) \,/\, (nS_{xx}-S_xS_x) \end{aligned}$$

où $S_x = \Sigma\,x_i$, $S_y = \Sigma\,y_i$, $S_{xx} = \Sigma\,x_i^2$ et $S_{xy} = \Sigma\,x_i\,y_i$, sommes calculées avec i (entier) variant de 1 à n.

Dans certains cas, on peut ramener la fonction étudiée à une relation linéaire. C'est possible avec une fonction logarithme $y = b + \mathrm{Log}\,x$ ou exponentielle $y = b\exp(ax)$ qui se prêtent à la méthode ci-dessus. Le cas fréquent d'un ajustement à une gaussienne $y = b\,\exp\,[(x-c)/a]^2$ se ramène également à une relation linéaire si le décalage c est connu. Sinon, on ramène y à une relation du second degré qu'on peut traiter comme le polynôme général ci-après.

En effet, bien souvent, si on ne peut faire mieux, on se rabattra sur une approximation par un polynôme général de degré m ,

$$f(x) = a_0 + a_1\,x + a_2\,x^2 + a_3\,x^3 + ... + a_m\,x^m$$

Une fonction définie par n points $\{x_i, y_i\}$ donnera n équations $f(x_i) = y_i$, système qui possède une solution et une seule si son déterminant est non-nul. Cependant, le polynôme trouvé, passant rigoureusement par les n points donnés (dans la mesure où les imprécisions de calcul sont nulles), est souvent trop complexe. Si ces points résultent de mesures, le polynôme comporte des détails ne traduisant que du bruit ou des erreurs accidentelles sans intérêt. On cherchera alors un polynôme de degré m inférieur à n. Cela s'appelle un *lissage* : le polynôme ne passera pas exactement par les points, mais s'en approchera au mieux. On peut faire varier m jusqu'à obtenir un compromis satisfaisant. On appliquera la méthode générale des moindres carrés qui optimise chaque paramètre a_i

(1) Ces calculs sont longs. Mais si n est égal à 2^k, k entier, on pourra les réduire par la technique de la transformation de Fourier rapide ou *FFT* (gain en temps $\approx n / \log_2 n$, soit un facteur 100 si n=1024. Cf. §11, page 170).

inconnu en égalant à 0 la dérivée partielle $\partial \mathbf{S}/\partial a_\mathbf{i}$. On obtient les *m* équations suivantes (avec $0<k<m$) :

$$\partial \mathbf{S}/\partial a_\mathbf{k}=0 \quad \Rightarrow \quad \sum \varepsilon_\mathbf{i}\, x_\mathbf{i}^\mathbf{k} = 0 \,, \qquad \text{soit}$$

$$\sum_{j=0}^{m} a_\mathbf{j} \sum_{i=0}^{n} x_\mathbf{i}^{\mathbf{k+j}} = \sum_{i=0}^{n} y_\mathbf{i}\, x_\mathbf{i}^\mathbf{k} \qquad \text{ou bien} \qquad \sum_{j=0}^{m} A_\mathbf{kj}\; a_\mathbf{j} = \sum_{i=0}^{n} y_\mathbf{i} x_\mathbf{i}^\mathbf{k}$$

Ces *m* lignes forment un système de *m* équations à *m* inconnues. Ces inconnues, les coefficients $a_\mathbf{j}$, s'obtiennent en résolvant le système, c.-à-d. en inversant la matrice des termes $A_\mathbf{kj}$ comme on a appris à le faire §4 :

$$a_\mathbf{k} = [A_\mathbf{kj}]^{-\mathbf{1}} \,.\, \sum y_\mathbf{i}\, x_\mathbf{i}^\mathbf{k}$$

D'autres procédés peuvent être utilisés comme celui des polynômes orthogonaux, qui améliore encore le résultat (on consultera des ouvrages spécialisés).

On peut également approcher une fonction de points à l'aide d'une suite d'arcs de courbe du 3e degré, s'appuyant sur les points en question et se raccordant les uns aux autres, ainsi que leurs dérivées, à chaque extrémité. C'est une fonction *spline*, proche de la forme qu'adopte une règle métallique déformable lorsqu'un dessinateur la force à suivre une courbe représentée sur le papier.

Une suite de fonctions de Bézier peut également servir d'approximation à une suite de points, si on veut se limiter au degré 3 : elles intéressent chacune 4 points $x_\mathbf{1}$, $x_\mathbf{2}$, $x_\mathbf{3}$, $x_\mathbf{4}$, passent seulement par les extrémités $x_\mathbf{1}$ et $x_\mathbf{4}$ et, en ces points, sont tangentes aux segments $x_\mathbf{1}\, x_\mathbf{2}$ et $x_\mathbf{3}\, x_\mathbf{4}$. Pour les raccorder sans point anguleux, il faut utiliser le point médian d'un triplet comportant trois points alignés, ou bien, plus généralement, créer des points intermédiaires pour obtenir de tels triplets.

6 - Résolution d'équations différentielles

Les procédés employés dérivent de la formule de Taylor. On progresse par pas, dans lesquels on assimile la fonction cherchée à sa tangente, ou mieux à son gradient, car on a souvent affaire à plusieurs variables. La méthode de **Runge-Kutta** est peut-être la plus utilisée : elle résout une équation différentielle du type

$$\mathrm{d}y/\mathrm{d}x \;=\; f\,(x,y)$$

par des itérations en *x* de pas Δx, donnant à *y*, au pas *i*+1, la valeur

$$y_\mathbf{i+1} \;=\; y_\mathbf{i} \;+ (\delta_\mathbf{1} + 2\delta_\mathbf{2} + 2\delta_\mathbf{3} + \delta_\mathbf{4})\; \Delta x/6$$

avec

$$\begin{aligned} \delta_\mathbf{1} &= f\,(x_\mathbf{i}, y_\mathbf{i}) \\ \delta_\mathbf{2} &= f\,(x_\mathbf{i} + \tfrac{1}{2}\Delta x, y_\mathbf{i} + \tfrac{1}{2}\delta_\mathbf{1}) \\ \delta_\mathbf{3} &= f\,(x_\mathbf{i} + \tfrac{1}{2}\Delta x, y_\mathbf{i} + \tfrac{1}{2}\delta_\mathbf{2}) \\ \delta_\mathbf{4} &= f\,(x_\mathbf{i} + \Delta x, y_\mathbf{i} + \delta_\mathbf{3}) \end{aligned}$$

Naturellement, dans ce problème, on doit connaître les conditions initiales, qui donneront les premières valeurs $x_\mathbf{0}$ et $y_\mathbf{0}$ pour i=0.

7 - Problèmes à N corps

On entend par *problème à 2 corps* l'étude du mouvement de deux astres soumis à la loi de la gravitation universelle. Ce problème a été résolu par Képler. Mais on ne sait pas résoudre le cas de 3 corps et plus (Poincaré a même démontré qu'il n'y avait pas de solution générale). On peut en trouver des solutions approchées, si l'influence d'un des astres sur les autres est faible, grâce à la *méthode des perturbations*, due à Laplace. C'est elle qui a permis la découverte de Neptune (par Le Verrier) et de Pluton. Elle est utilisée dans beaucoup d'autres cas, comme dans l'étude de la trajectoire des satellites artificiels (dans leur cas, la Terre ne peut d'ailleurs plus être assimilée à une masse ponctuelle).

Le problème à *N* corps peut être traité *numériquement* en progressant par petits intervalles de temps Δt pendant lesquels rien n'est censé *bouger* ; pour chaque corps, on calcule les forces exercées sur lui par les *N*–1 autres corps, puis on le déplace d'une petite quantité, calculée comme résultant d'un mouvement uniformément accéléré pendant Δt ; à partir des nouvelles positions, on calcule les nouvelles forces, etc... Le problème est résolu par *maillage*, comme dans le cas des équations différentielles.

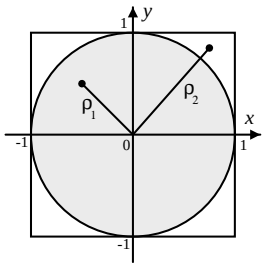
Il arrive souvent qu'on étudie l'interaction de *N* corps sans vouloir connaître leur mouvement complet, mais seulement leur position de repos (solutions d'équilibre stable, correspondant à des minimum d'énergie pour le système étudié). Le problème est moins détaillé que celui du mouvement des astres et on peut travailler sur un bien plus grand nombre de corps. Ces méthodes sont utiles en physique corpusculaire, atomique, structurale,... les corps en jeu étant les constituants de la matière.

8 - Hasard et calcul numérique

Certains problèmes peuvent exiger l'emploi d'une variable aléatoire. Les logiciels de programmation possèdent une fonction générant un nombre au hasard (compris entre deux limites). Si on la sollicite plusieurs fois, les quantités obtenues paraissent bien sans rapport entre elles. Cependant, si on reprend plus tard le même problème, on sera surpris de retrouver la *même séquence* de nombres prétendument aléatoires. Pour éviter cela, on peut parfois réinitialiser ladite fonction à une valeur elle-même "aléatoire". De toutes façons, les nombres produits sont fournis par un procédé déterministe, mais suffisamment complexe pour qu'il apparaisse aléatoire : par exemple, les décimales de π peuvent paraître distribuées au hasard, bien qu'elles soient parfaitement prédictibles.

On résout certains problèmes par une méthode faisant appel au hasard si on peut les ramener à l'étude d'une *densité de probabilité*. On effectuera une série de calculs en donnant aux variables initiales des valeurs aléatoires. Tel est en gros le principe de la **méthode de Monte-Carlo**, très utilisée en calcul numérique de haut niveau. Par exemple, on peut calculer le rapport entre deux aires comme la probabilité pour un point choisi au hasard de *tomber* dans l'une plutôt que dans l'autre.

Ainsi, on peut évaluer π comme limite du rapport $4p/q$ dans l'*expérience* suivante renouvelée q fois : on prend deux nombres x et y au hasard, mais compris entre -1 et 1, et on compte le nombre p de cas où la distance (au carré) $\rho = x^2+y^2$ est inférieure à 1 ; le point est alors dans le cercle, cf. figure ci-contre (calcul de π peu rapide).



9 - Frontières du déterminisme

Si l'analyste essaie parfois d'injecter du hasard dans le déterminisme du calculateur, celui-ci, en revanche, peut introduire le *chaos* là où on s'attendrait à un déterminisme rigoureux. Ce problème n'est pas vraiment du ressort de l'informatique, mais c'est elle qui l'a soulevé de manière aiguë. On a trouvé là une limite quasi-théorique qui a déçu bien des espoirs dans la toute-puissance du calcul numérique.

Il en est ainsi souvent lorsqu'on cherche une loi d'évolution pour un problème non-linéaire. On trouve bien sûr une solution, mais en l'approfondissant, on s'aperçoit que la loi trouvée n'est pas *stable* : les différentes solutions du même problème, calculées pour des paramètres voisins (des *conditions initiales* voisines), vont diverger les unes des autres d'une quantité inhabituelle pour des problèmes déterministes (cf. §3 à la fin, où le passage de la constante a de 3 à un peu plus de 3 change totalement le résultat).

Les météorologues sont particulièrement confrontés à ce comportement. On avait cru qu'on obtiendrait des prévisions de plus en plus précises en affinant le maillage de l'atmosphère, en multipliant les mesures dans chaque maille, en améliorant les équations décrivant les échanges d'énergie... Or certaines de ces équations sont non-linéaires et leurs solutions se comportent vite de façon quasi-chaotique. La prévision météo à long terme (c'est-à-dire au delà de 5 à 6 jours) semble se heurter là à une limite infranchissable.

10 - Chaos et fractales

Ce comportement chaotique apparaît également lorsqu'on étudie un phénomène décrit par une relation de **récurrence** comportant un terme non-linéaire (cf. §3, page 166). En voici un exemple célèbre qui permettra de créer sur l'écran des dessins étranges et envoûtants. La loi d'itération très simple,

$$z_{i+1} = z_i^2 + c$$

où z et c sont des nombres complexes **(2)**, génère les **fractales de Julia** avec c=cte, si on porte l'image de z dans le plan complexe **(3)** après un grand nombre d'itérations. Si c est nul (ou a et b), la position finale de z dépend de la valeur initiale de son module ρ_0 (tel que $\rho_0^2 = x_0^2+y_0^2$). Si $\rho_0<1$, le point converge vers l'origine (0,0) ; on dit qu'elle est pour lui *un attracteur* ; si $\rho_0>1$, le point diverge à l'infini ($\rho_i \to \infty$) ; enfin, pour tout $\rho_0 = 1$, le point reste constamment sur le cercle Γ de rayon unité centré en (0,0). Ce cercle est appelé *séparatrice*, car il sépare les régions de convergence et celles de divergence.

Si c n'est plus nul, le comportement de z devient surprenant. On obtient plusieurs attracteurs avec des séparatrices aux formes étranges, brisées et ramifiées

(2) le lecteur non initié aux complexes traduira cette écriture par le système réel : $x \leftarrow x^2 - y^2 + a$ et $y \leftarrow 2xy + b$ (a et b constants).

(3) i.e. si on représente y en fonction de x dans un plan cartésien réel.

à l'infini. Ces courbes sont des **fractales**. Si l'on veut les détailler (avec des pas plus petits en *x* et *y)*, on leur trouve toujours la même allure (elles sont dites *autosimilaires*). Leur structure n'est pas sans rappeler la forme des cristaux, des végétaux, des roches, des montagnes, des rivages ... et celles des multiples *agrégats* naturels (flocons de neige, colloïdes, aérosols, macromolécules ...). Si on remplace la constante *c (*ou *a* et *b)* par une relation plus complexe, on obtient des dessins différents, mais tout aussi fascinants.

Ces fractales ont été étudiées récemment par **Mandelbrot** qui a montré l'importance fondamentale de l'ensemble portant son nom (EM), ensemble des points construit comme ci-dessus, mais représentant maintenant les séparatrices dans l'itération en *z* en fonction des coordonnées variables *a* et *b* avec $z_0 = 0$.

On programme aisément le dessin de l'ensemble de Mandelbrot par une triple boucle. Les deux boucles externes définiront les coordonnées *i*, *j* d'un point de l'écran ($1 \le i \le n$ et $1 \le j \le m$, *n* et *m* étant égaux aux nombres de pixels en horizontal et en vertical). La boucle interne effectue l'itération en *x* et *y* , chacune de ces variables étant nulle au départ. On comptera le nombre de pas *K* nécessaires pour que ρ devienne grand ($\rho_K >> 1$) et on donnera au point *(a,b)* ou *(i,j)* une couleur fonction de *K* (couleur d'isostabilité). On mettra toutefois une limite à l'itération et on coloriera en noir les points n'ayant pas divergé ($\rho_K < 1$) durant ce laps de temps. Patience, car ces calculs sont longs.

Un ensemble de Julia se programme de manière analogue, selon le tableau ci-après. La constante *c* qui le caractérise fait partie de l'EM. Pour obtenir les plus *beaux* ensembles de Julia, prendre des valeurs de *c* situées sur la séparatrice, surtout celles *nichées* au creux de ses *bourgeons*. On s'apercevra qu'une très faible différence sur les conditions initiales *a* et *b* peut changer totalement la figure. C'est là une des caractéristiques des fractales et du comportement chaotique.

Ensemble :	**de Mandelbrot**	**de Julia**
Itération	$z = z^2 + c$	$z = z^2 + c$
ou bien	$x = x^2 - y^2 + a$	$x = x^2 - y^2 + a$
et	$y = 2xy + b$	$y = 2\,xy + b$
Valeur départ de *x* et *y*	0, 0	x_0 et y_0
Coordonnées écran	*i* et *j*	*i* et *j*
Coord. algébr., horizale	$a = a_1 + i(a_2 - a_1)/n$	$x_0 = x_1 + i(x_2 - x_1)/n$
verticale	$b = b_1 + j(b_2 - b_1)/m$	$y_0 = y_1 + j(y_2 - y_1)/n$
Valeurs suggérées	$a_1 = 2{,}25$; $a_2 = 0{,}75$ $b_2 = -b_1 = 1{,}5$	$x_1 = y_1 = -x_2 = -y_2 = 1{,}4$ $a = -0{,}12$; $b = 0{,}74$

11 - Note sur la transformation de Fourier rapide (TFR, FFT).

La TFR consiste à calculer par 3 boucles imbriquées les composantes réelle et imaginaire $F_r(k)$ et $F_i(k)$, TF de la fonction réelle *f(i)* donnée sur *N* points, si $N = 2^m$, *m* entier.

– La boucle externe, $0 \le k < m$, calcule les entiers $n_i = 2^{k-1}$ et $n_j = N/(2n_i)$, puis régénère les tableaux de travail $t_r(i) = F_r(i)$ et $t_i(i) = F_i(i)$ pour $0 \le i < N$;

– La boucle médiane fait varier *j* : $0 \le j < n_j$;

– La boucle interne, $0 \le i < n_i$, calcule les indices $p = i + 2\,j\,n_i$ et $q = p + n_i$, les fonctions $c = \cos(a\,i\,n_j)$ et $s = \sin(a\,i\,n_j)$, avec $a = 2\pi/N$ (il vaut mieux dresser la table des *N*/2 valeurs de ces fonctions avant le début du calcul), et enfin

$$\begin{aligned} F_r(p) &= t_r(p) + c.t_r(q) - s.t_i(q) \\ F_i(p) &= t_i(p) + s.t_r(q) + c.t_i(q) \\ F_r(q) &= t_r(p) - c.t_r(q) + s.t_i(q) \\ F_i(q) &= t_i(p) - s.t_r(q) - c.t_i(q) \end{aligned}$$

Avant départ, il faut égaler le tableau F_i à 0 et F_r à celui des données *f(i)*, mais **après sa réorganisation**. Le tableau initial $F_r(i) = f(j)$ ($0 \le j < N$) où *j* est un entier de *m* bits symétrique (ou *miroir*) de *i* en binaire (Ex. avec *N*=16 : *i*=8 → *j*=1, 10→5, 1→8... Avec *N*=1024 : 512→1, 256→2, 768→3... Opérer en divisant *i* et en multipliant *j* par 2, *m* fois, avec transfert sur *j*, à chaque fois, du bit débordant de *i*)**(4)**.

Cette méthode repose sur la décomposition des *f(i)* en deux séries respectivement sur les indices pairs et impairs pour les *N*/2 premières valeurs de *F(k)*, les suivantes se déduisant des premières. On réitère cette répartition *m* fois, prélevant ainsi les données par un jeu de *peignes* imbriqués avec des dents de plus en plus espacées, jusqu'à obtenir *N* peignes d'une seule dent. Cette transformée ne comprend que des valeurs de la fonction *f* d'origine et ce, quel que soit *k*, mais à condition que *f* ait été réorganisée (en prenant à la place de f_i le terme f_j dont l'indice *j* est le miroir binaire de *i*, comme indiqué ci-dessus).

Les passages d'un peigne à l'autre intéressent chaque fois toute une famille de termes *k* combinés et on économise ainsi beaucoup de temps, la TF étant calculée de manière globale et non plus terme à terme.

(4) Si $j = \sum_{k=0}^{r=m-1} 2^k b_k$, son miroir est $i = \sum_{k=0}^{r} 2^{r-k} b_k$, $b = 0$ ou 1

SOLUTIONS DES EXERCICES

Page 13

n°1 : 1 0001 0000, 1001 1010, 1 0101 1011, 1 0000 0011, 110 1000, soit 9B+75=110_{16}, 37+63=9A, FE+5D=15B, 54+AF=103, 4D+1B=68

n°2 : 010 0110, 101 0100, 1010 0001, 0010 0101, –1000 1010,
ou bien 9B–75=26, B7–63=54, FE–5D=A1, D4–AF=25, 4D–D7=–8A

n°3 : 2^{10}=1024≈10^3, 2^{20}≈10^6, 2^{30}≈10^9 (cf. p. 103, §4)

n°4: $1633C_{16}$, FC0F, 18325_{16}, 7A82, 7D8E, 8918_{16}

n°5: 7A82 = 0111 1010 1000 0010 = 0 111 101 010 000 010 = 75202_8
E8BA = 1110 1000 1011 1010 = 1 110 100 010 111 010 = 164272_8
75102_8 = 111 101 001 000 010 = 111 1010 0100 0010 = 7A42
167272_8 = 001 110 111 010 111 010 = 0 1110 1110 1011 1010 = EEBA

n°6, 1ère question : bien que *a* et *b* soient inversés (0 et 1 permutés), la table de vérité de $\overline{a}$ XOU $\overline{b}$ est identique à celle de *a* XOU *b*, page 24.

2e question : Les propositions *a* et *b* étant équiprobables, le seront également les événements représentés par chaque case des tables de vérité. La probabilité d'un événement étant égale au rapport entre le nombre de cas favorables et celui de cas possibles, on déduit des tables de vérité les pourcentages indiqués.

n°7 :

a	XOU	b			XOU	b		
0	XOU	0	→	0	XOU	0	→	0
		1	→	1	XOU	1	→	0
1	XOU	0	→	1	XOU	0	→	1
		1	→	0	XOU	1	→	1

= a

Page 23.

n°1 : A la 16e impulsion, on retrouve le compteur dans le même état qu'avant la première. Il compte donc jusqu'à 16, c.-à-d. 2^4, ce qui est normal puisqu'il est formé de 4 bascules.

n°2 : RAM = mémoire vive, ROM = mémoire morte. Toutes deux sont d'accès direct (ou aléatoire). Les SRAM sont faites de bascules, les DRAM, plus simples, comportent malgré tout un transistor, comme les EEPROM et les REPROM, mais les ROM au sens strict et les PROM n'en ont pas.

n°3 : A la 15e impulsion, les sorties S2 des 4 bascules sont à "1", la 16e les faisant toutes passer à 0. Pour limiter le compteur à 10, on ajoute une porte qui sera ouverte par B4 lors de son état haut (8e impulsion). A la 9e, cette porte laissera passer le front de basculement de B1 qui forcera à "1" B2 et B3. Ainsi, dès ce nombre, le compteur est dans l'état qu'il aurait à la 15e impulsion.

n°4 : Le *compteur* EDF **mesure** une énergie, le *compteur* à eau **mesure** une quantité d'eau, le tachymètre d'un véhicule **mesure** une vitesse. Seuls les compteurs de taxes téléphoniques et ceux sur la chaussée **comptent** des objets *discrets* (grâce à des impulsions électriques ou mécaniques).

n°5 : Dans ce cas, la porte est un circuit ET. Il arrive qu'on appelle aussi *porte* des circuits $\overline{\text{ET}}$ ou même des circuits OU.

Page 31.

n° 1 et 2 : voir texte.

n° 3 : L'adressage est appelé complexe quand on veut "adresser" un volume de mémoire supérieur à la capacité du bus: 2^8 = 256 ou 2^{16} = 65536. Il faut alors adresser en deux temps, en considérant que la mémoire est formée de pages d'un volume au plus égal à la capacité du bus.

n°4 : voir texte.

n°5 : Dans un transfert d'information, l'UC place sur le bus-adresse l'adresse de l'élément avec qui elle veut échanger, puis excite dans le bus-commande la ligne définissant le type d'échange (lecture, écriture, entrée, sortie). Le circuit émetteur (UC ou module adressé) place l'information sur le bus-données.

n°6 : cf. pages 67-69.

n°7 : cf. page 59 (chaque bit est indépendant et joue un rôle spécial).

n°8 : cf. texte

Page 41. Les mémoires volatiles sont celles de type RAM, formées de transistors et-ou de condensateurs à décharge rapide. Ni les mémoires magnétiques, ni les ROM ne sont volatiles, car elles ne comportent ni transistor, ni condensateurs. Les EPROM en comportent, mais leur condensateur se décharge très lentement.

Pages 51.

n° 1 : Avec environ 70 caractères par ligne, 50 lignes par page et 285 pages, le texte de ce livre atteint le mégaoctet, ce qu'une disquette haute densité contient aisément. Cependant, les figures et les codes de présentation (cf. traitement de texte) doublent à peu près ce volume.

n°2 : Pour *l'Encyclopédie Universelle*, avec un nombre de $20 \times 1100 \times 3 \times 100 \times 44$ caractères, soit 290 Mo, il faudrait plus de 200 disquettes (toujours sans prendre en compte les figures). Mais un seul disque optique serait suffisant.

n°3 : Si on admet que 25 ans séparent deux générations humaines, il n'y en a eu que 100 000 depuis l'aube de l'Humanité, nombre juste supérieur à la capacité d'un entier simple, mais très inférieur à celle d'un entier long ou d'un réel.

n°4 : Durée du trajet de la lumière entre Terre et α-Centauri: $4{,}3 \times 365 \times 24 \times 3600 = 1{,}4\ 10^{8}$ s, soit $1{,}4\ 10^{23}$ fs. Distance Terre-α-Centauri: $1{,}4\ 10^{8} \times 3\ 10^{8}$ m = $4\ 10^{16}$ m = $4\,10^{31}$ fm. Tous ces nombres ne dépassent pas la capacité des flottants simples, tant en ordre de grandeur qu'en précision.

n°5 : Age de l'Univers: $15\ 10^{9} / 10^{-15} = 1{,}5\ 10^{25}$ fs, nombre largement inférieur à la capacité du flottant simple.

n°6 : Nombre d'atomes : $1{,}99\,10^{30} \times 1000 / 332\,300 / 4{,}7\,10^{-23} = 1{,}3\,10^{50}$ environ pour toute la Terre. Ce nombre est très inférieur à la capacité du flottant de double précision, mais, à cause de son exposant, il ne *tient* pas dans un flottant simple. Bien que se rapportant à des objets de mêmes tailles, ce nombre est supérieur aux précédents parce qu'il traduit un rapport entre des *volumes* et non entre des *longueurs*. Il sera donc de l'ordre du cube des précédents.

n°7 : Masse d'or dans l'océan : $4\,10^{-6} \times 1370\,10^{6} \times 10^{9} \times 1{,}035 = 5{,}7\,10^{12}$ g, soit 5700 kilotonnes. Ce nombre de faible précision (deux chiffres) *tient* aisément dans un entier simple si, après calcul des exposants à part, on choisit pour l'exprimer une unité adéquate, la kilotonne.

Page 60. **n°1** : peu de différences de fond (voir texte).

n°2, 3, 4 : voir texte.

n°5 : Les sauts et les appels de SP.

n°6 : cf. pp. 111 et 134. Une correction s'impose sur chaque quartet, puisqu'une partie de leur plage de représentation est interdite.

n°7 et 8 : voir texte.

n°9 : trois moyens courants: registres, pile et variables globales (ou communes). Autre possibilité: échange de fichiers (non citée dans le livre).

n°10 : Cette instruction sert à remettre un registre à zéro car a XOR $a = 0$, $\forall\ a$.

n°11:

```
MOV AX COEFB ;      b en AX
MOV BX AX ;         b copié en BX
PUSH AX ;           b dans la pile
MUL AX BX ;         b2 en AX
PUSH AX ;           b2 dans la pile
MOV AX COEFA ;      a en AX
MOV BX COEFC ;      c en BX
MUL AX BX ;         ac en AX
SHL AX,2 ;          ac décalé donne 4 ac.
NEG AX ;            -4ac
POP BX ;            b2 en BX
ADD AX,BX ;         b2-4ac=D
JSN dneg ;          saut si D<0
JZ  dzer ;          saut si D=0
CALL RAC ;          racine en AX
MOV CX, AX ;        copie
.....
```

Pages 81-82.

n°1: $j * (i/j) + i \text{ MOD } j = i$ (découle de la définition de l'opérateur MOD).
n°2: temps = 6 h, 11 mn, 9,9 s.
n°3: les trains se croisent en t=13h 12 mn 10 s à x=336,8 km

Pages 93-94.

n°1: Recherche de sous-chaînes.

Le document à corriger est censé être constitué par le fichier C:\doc\text qu'on ouvrira deux fois, la première servant à déterminer sa longueur, et qu'on placera dans la chaîne txt$. Les chaînes de plus de 250 octets n'étant pas permises en Basica, il faudrait dans ce cas ne lire le fichier que par environ 230 caractères à la fois. Les espaces ajoutés ligne 50 servent à résorber les décalages introduits lors du doublement éventuel des lettres n.

```
10   DEFINT A-S
20   CLS: OPEN "C:\doc\text.doc" FOR APPEND AS #1
30   l = LOF(1):     CLOSE #1
40   OPEN "C:\doc\texthon.doc" FOR INPUT AS #1
50   txt$ = INPUT$(l - 1, #1) + "                    "
60   PRINT txt$:
100  j = 1
110      WHILE j > 0: j = INSTR(j + 1, txt$, "hon")
115          IF j > 0 THEN GOSUB 500:
120      WEND:       j = 1:
130      WHILE j > 0: j = INSTR(j + 1, txt$, "HON")
135          IF j > 0 THEN GOSUB 500:
140      WEND:       j = 1:
150      WHILE j > 0: j = INSTR(j + 1, txt$, "Hon")
155          IF j > 0 THEN GOSUB 500:
160      WEND
180  PRINT : PRINT txt$
200  END                 '   Fin du principal
300      ' .  Sous-programme de traitement
500      FOR i = j TO j - 25 STEP -1: c$ = MID$(txt$, i, 1)
510        IF ASC(c$) < 65 THEN GOTO 600   '
                                  recherche début de mot
520      NEXT i
530  PRINT  "Anomalie en position "; i; " --> ";
                                  MID$(txt$,i,25)
600  FOR k = j TO j + 25: c$ = MID$(txt$, k, 1)
610      IF ASC(c$) < 65 THEN GOTO 700   ' rech. fin mot
620  NEXT k
630  PRINT  "Anomalie en position "; i; " --> ";
                                MID$(txt$,i,25)
700  PRINT  "En";i;" --> "; MID$(txt$, i+1, k-i-1); TAB(30);
710  PRINT  "[ +, -, RC ]"; :
720  r$= "":  WHILE r$ = "":  r$=INKEY$:  WEND
                                              'attente frappe
750  IF ASC(r$) = 13 THEN PRINT : RETURN
760  IF ASC(r$) = 27 THEN 180
810  IF r$ = "-" THEN 950
900  IF r$ <> "+" THEN PRINT : BEEP: GOTO 700
910  MID$ (txt$, j+4, l - j-3) = MID$(txt$, j+3, l - j-3)
920  MID$ (txt$, j+3, 1) = "n":    l = l+1          ' insertion
930  PRINT "  -->  "; MID$(txt$, i + 1, k - i)
940  RETURN
950  MID$ (txt$, j+2, l-j-3) = MID$(txt$, j+3, l-j-3)
970  MID$ (txt$, l - 1, 1) = " "
980  PRINT "  -->  ";  MID$(txt$,i+1,k-i-2)      ' suppression
990  RETURN
```

n°2: Tri d'une liste de nombres réels en désordre.

Elle est également censée être contenue dans un fichier.

```
10   DEFINT I-M, S
20   CLS : l = 0: PRINT "Liste initiale :"
30   OPEN "c:\doc\listdeso.doc" FOR INPUT AS #1
40       WHILE NOT EOF(1)
50       INPUT #1, var: l = l + 1:  PRINT var; "  ";
60       WEND:          CLOSE #1:   DIM v(l)
70   OPEN "c:\doc\listdeso.doc" FOR INPUT AS #1
80   FOR i = 1 TO l: INPUT #1, v(i): NEXT i
100  stab = 0
110  FOR i = 1 TO l-1
120      IF v(i) < v(i+1) GOTO 140
130      stab = 1: SWAP v(i), v(i+1)
140  NEXT i
200  IF stab=1 GOTO 100
300  PRINT : PRINT : PRINT "Liste ordonnée : "
310  FOR j = 0 TO l STEP 8
320      FOR i=1 TO 8:
         IF j+i <= l  THEN PRINT v(j+i);TAB(9*i);
330      NEXT i: PRINT
340  NEXT j
350  END
```

n°3: Calcul des nombres premiers jusqu'à 1000

```
10          '    Nombres premiers
20    DEFINT I-M
30    CLS :
40    PRINT "Nombres premiers compris entre 2 et 1001 :"
50    FOR i = 3 TO 1000
60        FOR j = 2 TO i/2:  IF i MOD j=0 GOTO 90
70        NEXT j
80    PRINT i ;              ' nb. premier si sortie normale
90    NEXT i
```

n° 4: Calcul de la racine carrée d'un nombre positif

```
10  CLS :   prec# = 1E-15
20  INPUT "Donnez un nombre (ou tapez 'fin') : ", rep$
30  IF LEFT$(rep$, 1) = "F" OR LEFT$(rep$, 1) = "f"
            GOTO 99
40  nb# = VAL(rep$):      rac# = nb#/2#:      dif# = nb#/4
50  IF nb#<0 THEN PRINT " --> Impossible: nb. négatif":
            GOTO 20
60      WHILE ABS(dif#) > prec#
70      rac# = rac# + nb# * dif# / (2 * rac#)
75      dif# = 1 - rac# * rac# / nb#
80      WEND
90  PRINT "racine = ";
        USING "######.################"; rac#
91  PRINT "SQR(x) = ";
        USING "######.################";  SQR(nb#)
95  PRINT :  GOTO 20
```

n°5 et **6:** Calcul de π

Les 3 SP (en 100, 200, 300) traduisent les 3 méthodes exposées pages 234-235. On vérifiera que la seconde ne donne pas un résultat précis. On quitte par la touche **Echappement** *(code ASCII 27) ou par les lettres* **F** *ou* **f** .

```
5     DEFINT I-M, T
10    CLS : INPUT "Type de relation (1,2,3) ? ", TYP
15    F$ = "##.#^^^^"
20    CLS : PRINT "Type"; TAB(6); "Pas"; TAB(16); "N";
                TAB(27); "S.int";
25    PRINT TAB(51); "S.ext"; TAB(71); "Diff"
30    i = 0: N# = 4: a# = SQR(2#): b# = 2#
35        IF TYP = 1 THEN GOSUB 100
40        IF TYP = 2 THEN GOSUB 200
45        IF TYP = 3 THEN GOSUB 300
50    PRINT TYP; TAB(5); i; TAB(12);
            USING "##########"; N#;
60    PRINT TAB(24); N# * a#; TAB(48); N# * b#;
70    PRINT TAB(69); USING F$; N# * (a# - b#)
75    R$ = "": WHILE R$ = "": R$ = INKEY$: WEND
                                    ' attente
80    IF ASC(R$) = 27 OR LEFT$(R$, 1) = "F"
          OR LEFT$(R$, 1) = "f" THEN END
85    i = i + 1: N# = N# * 2
90    GOTO 35
                              ' sous-programmes
100   b# = a# * b# / (a# + b#):
      a# = SQR(a# * b# / 2#):    RETURN
200   a# = SQR(2# - SQR(4# - a# * a#)):
250   b# = (2# / b#) * (SQR(4# + b# * b#) - 2#): RETURN
300   a# = a# / SQR(2# + SQR(4# - a# * a#)):
350   b# = 2# * b# / (2# + SQR(4# + b# * b#)): RETURN
```

Valeur de π avec 20 chiffres : 3,1415 92653 58979 32384..
La relation entre les côtés b_{2n} et b_n des polygones circonscrits est, sauf erreur :

$$b_{2n} = \left(-4 + 2\sqrt{4 + b^2_n} \right) / b_n$$

INDEX

NOTE : *Dans l'index qui suit, ne sont répertoriés que les mots propres à l'informatique et ceux ayant dans ce livre un sens spécial. Les numéros donnés sont ceux des pages. S'ils sont suivis d'un astérisque, ils renvoient à une note en bas de page. Les numéros reliés par un tiret se réfèrent à une suite continue de pages (ajouter parfois au numéro suivant les premiers chiffres du numéro initial). Une suite de points signale un grand nombre de références successives moins importantes.*

ABREVIATIONS UTILISEES

Dans le livre :

AC	:	avant Jésus-Christ
c.-à-d.	:	c'est-à-dire
ch., chap.	:	chapitre
cf.	:	se reporter à
fig.	:	figure
i.e.	:	id est, c'est-à-dire
LM	:	langage machine, numérique ou
	:	symbolique ("assembleur")
o, ø, Ø	:	octet
p., pp.	:	page, pages
par ex.	:	par exemple
PC	:	calculateur ou ordinateur personnel
PS	:	PostScript
SE	:	système d'exploitation
SGBD	:	système de gestion de base de données
TTX	:	traitement de texte
UC	:	unité centrale

Dans l'index :

coordon.	:	coordonnées
math.	:	mathématique
typo.	:	typographie

ainsi que des unités usuelles en physique : A (ampère), kg (kilogramme-masse), m (mètre), m/s (mètre par seconde), s/m (seconde par mètre), V (volt), Ω (ohm), leurs multiples et sous-multiples (voir page 5) et l'unité anglaise " (pouce ou inch, de valeur 25,4 mm).

POUR EN SAVOIR PLUS ...

Littéraire
Le défi informatique, Bruno LUSSATO, Fayard, 1981

Initiation
L'ordinateur et l'informatique en 15 leçons, P. MORVAN, Ed. Radio, 1977
Le (les, la) ... Comment ça marche ?, DUNOD (collection)

Historique
Ainsi naquit l'informatique, R. MOREAU, Dunod, 1981

Circuits de base
Logique et organes des calculatrices numériques, G. BOULAYE, Dunod, 1970
Du composant au système, introduction aux microprocesseurs, Rodnay ZAKS, Sybex, 1981
Pratique des circuits logiques, J-M. BERNARD et J. HUGON, Eyrolles, 1985

Théorie de l'information, codages, principes
L'automatique des informations, F-H. RAYMOND, Masson, 1957
Eléments fondamentaux de l'informatique, P. POULAIN, Dunod, 1967

Langages machines
Calculatrices de poche et informatique, P. VITRANT, Masson, 1980
Comprendre la compilation, P.Y. CUNIN, M. GRIFFITH, J. VOIRON, Springer-Verlag, 1980
Programmation du 8086-8088, James W. COFFRON, Sybex, 1983

Systèmes d'exploitation
L'exploitation partagée des calculateurs, J. BERTIN, M. RITOUT, J.C. ROUGIER, Dunod, 1967
Unix, système et environnement, A.B. FONTAINE, P. HAMMES, Masson, 1984
Principes des systèmes d'exploitation des ordinateurs, S. KRAKOWIAK, Dunod, 1987
La Bible du PC, Michael TISCHER, Ed. Micro-Applications, 1988
PC interdit, B. BERTELSONS et M. RASCH, Ed. Micro-Applications, 1994
Linux par la pratique, Daniel A. TAUBER, Sybex, 1995 (logiciel sur disque optique inclus)

Langages évolués
Initiation au Basic, H. LILIEN, Ed. Radio, 1982
Le langage C, B.W. KERNIGHAN et D.M. RITCHIE, Masson, 1983
Pascal : *norme ISO et extension*, Patrice LIGNERET, Masson, 1983
Langage d'un autre type : LISP, Christian QUEINNEC, Eyrolles, 1984
Quick-C, Gérard LEBLANC, Eyrolles, 1988
ADA avec le sourire, JM. BERGER, L. Donzelle, V. OLIVE, J. ROUILLARD, Presses Polytech. Romandes, 1989

Bureautique
Bases de données et systèmes relationnels, C.DELOBEL, M.ADIBA, Dunod 1982
Micro-édition sur PC, PS et compatibles, J-P. LAMOITIER, M. TREILLET, B. CHANTALOU, Cédic-Nathan.

Dessin
PostScript, Language Reference Manual, Adobe Syst., Addison-Wesley, 1985
PostScript, l'essentiel, Pierre BLANC, Eyrolles, 1993

Télématique
Réseaux, architectures, protocoles, applications, A. TANNENBAUM, InterEditions.
Téléinformatique, C. MACCHI, J.-F. GUILBERT et al., Dunod, 1987
L'Internet professionnel, (ouvrage collectif), Editions du CNRS, 1995

Applications industrielles ou en temps réel
Ordinateurs en temps réel, applications industrielles, J.P. NANTET, Masson, 1970
Acquisition et traitement des données, H. SOUBIES-CAMY, Ed. Radio, 1970

Techniques avancées
Pratiquez l'intelligence artificielle, Jean-Pascal AUBERT, Eyrolles, 1983
L'homme FACe à l'intelligence artificielle, J.-D. WARNIER, les Editions d'Organisation, Paris, 1984
Réseaux de neurones, E. DAVALO, P. NAIM, Eyrolles, 1989

Société
Le pirate de l'informatique, Bill LANDRETH, Cedic-Nathan, 1985