# Introduction à la programmation

## Sommaire

# 1. QUELQUES DEFINITIONS

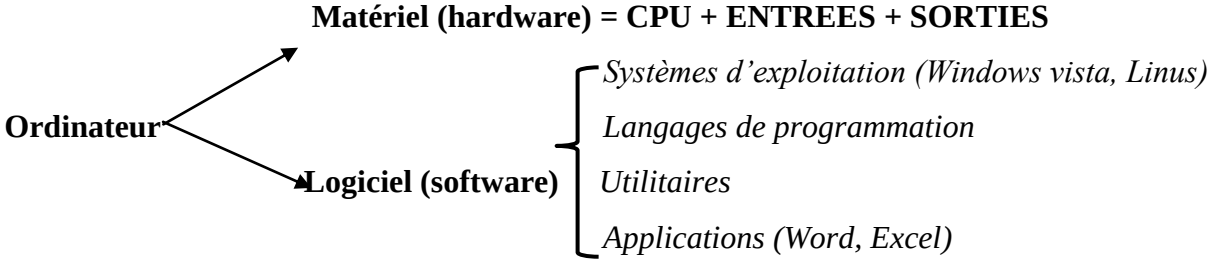
L'**informatique** est l'ensemble des sciences et techniques de traitement automatique de l'information. Selon Edsger Dijikstra, c'est la science des ordinateurs.

Un **ordinateur** étant une machine de traitement automatique de l'information, toute utilisation d'ordinateur nécessite deux éléments combinés :

- le **matériel** (partie physique, hardware en anglais) qui est constitué d'une unité centrale à laquelle sont reliés des périphériques d'Entrée/Sortie;
- le **logiciel** (partie intellectuelle, software en anglais) qui désigne l'ensemble des programmes utilisables pour traiter des informations avec le matériel. On en distingue entre autres les systèmes d'exploitation et les applications ; entre ces deux extrêmes, on inclut les langages de programmation et les utilitaires.

Un **programme** est une suite statique d'instructions permettant de réaliser une ou plusieurs tâches ou de résoudre un problème. Un programme est l'expression d'un algorithme dans un langage donné pour un matériel donné.

Un **algorithme** est une suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème (selon Petit Larousse) : c'est la description logique d'un programme.




Quoique notre ambition est de satisfaire la curiosité des débutants programmeurs, nous ne pourrons pas expliquer dans les moindres détails toutes les techniques de la programmation existantes, et ce pour des raisons d'efficacité. Nous allons dans les sections qui suivent nous attacher à exposer quelques éléments fondamentaux de la programmation structurée.

Notre souhait est que ce cours d'une part éveille chez les programmeurs débutants la passion de la programmation et d'autre part confirme les programmeurs déjà avertis dans leurs expériences.

Après plusieurs années d'expérience et d'échange avec un très large éventail de programmeurs et analystes-programmeurs et suite à la lecture de nombreux livres sur la programmation, nous avons déduit que les bonnes pratiques de la programmation ont pour source la maîtrise et la mise en application des concepts de la programmation. La discipline de la programmation, comme tout talent intellectuel, requiert un certain effort personnel pour être maîtrisée. A cet effet, nous citons :

- « L'école tue les génies »
- Jules Renard : « Au travail, le plus difficile, c'est d'allumer la petite lampe du cerveau. Après, ça brûle tout seul. »
- Ps 119:99 « Je suis plus instruit que tous mes maîtres, Car tes préceptes sont l'objet de ma méditation.»
- Thomas Edison : « Le génie est fait de dix pour cent d'inspiration et de quatre-vingt-dix pour cent de transpiration »
- Henri Kaiser disait : "Une minute de planning vous en épargne deux pour l'exécuter."
- « Votre travail reflète l'image de celui qui l'accomplit, aussi, ajoutez-y la signature de l'excellence ! » Bob GASS
- Jacques Lesourne : « les tâches d'un professionnel sont : s'informer, innover, concevoir, entreprendre, achever, etc. »
- Jean 13:17 «Si vous savez ces choses, vous êtes heureux, pourvu que vous les pratiquiez. »
- « L'informatique semble encore chercher la recette miracle qui permettra aux gens d'écrire des programmes corrects sans avoir à réfléchir. Au lieu de cela, nous devons apprendre aux gens comment réfléchir » Auteur anonyme

QUALITES D'UN PROGRAMMEUR

Il faut savoir comment un ordinateur fonctionne (cf. cours ATO : Architecture et Technologie des Ordinateurs, Systèmes d'exploitation, Algorithmique) pour comprendre ce qu'on fait.

Un programmeur doit avoir certaines qualités comme :

- **La patience** : un programme ne marche jamais du premier coup, il faut savoir persévérer!
- **Le sens de la logique** : pas besoin d'être fort en maths certes, mais ça ne vous empêchera pas d'avoir à réfléchir.
- **Le calme** : on ne tape pas sur son ordinateur avec un marteau. Ce n'est pas ça qui fera marcher votre programme.

# 2. QUELQUES MAXIMES A CONNAÎTRE DE TOUT PROGRAMMEUR

**Être clair pour être rigoureux**

Soyez clair et pas trop astucieux

Soyez rigoureux avant d'optimiser

Restez rigoureux en optimisant

Soyez clair avant d'optimiser

Restez simple pour optimiser

**Écrire pour être lu**

Polissez la forme pour aider le lecteur

Écrivez pour être lu de haut en bas

Commentez le code sans le trahir

Évitez l'excès de commentaires

Personnalisez et initialisez chaque variable

**Être cartésien**

Écrivez en clair un premier programme

Décomposez-le en blocs

Cantonnez chaque bloc à une action bien faite

Éclairez les liaisons entre bloc

Indentez pour exhiber la structure logique.

# 3. LES PHASES DE DEVELOPPEMENT DE LOGICIEL

*Phase = états successifs d'un phénomène en évolution.*

La programmation est l'ensemble des activités qui permettent d'élaborer des programmes. (C'est l'art de programmer, l'art de réaliser des "programmes informatiques")

L'élaboration de programme suit une démarche progressive qui peut être décomposée en phases comme suit :

- spécification (définition du problème dans un cahier de charges : entrées, sorties et traitements),
- conception (analyse en vue d'une solution meilleure),
- codage (écriture du programme dans un langage de programmation),
- documentation (rédaction des guides d'utilisateur et de maintenance),
- tests
- correction et mises au point,
- maintenance (mise à jour du programme).

La programmation au sens réduit est le terme qui couvre les travaux depuis l'étape de codage jusqu'à l'étape de maintenance.

### 4.1.Spécification du problème

Avant de commencer par élaborer un programme, quelle que soit sa simplicité, il y a intérêt à bien connaître le problème, il y a bien intérêt à définir le problème de telle manière qu'il puisse être analysé en vue d'une solution meilleure.

La définition du problème se fait dans un document appelé cahier des charges. Un cahier de charges comporte entre autres et d'une façon détaillée :

- les sorties : la liste des résultats que fournira l'ordinateur ;
- les traitements : la liste des traitements (formules, calculs, contraintes, contrôles à effectuer etc.) ;
- les entrées : la liste des données de départ et de référence.

Le cahier des charges est souvent la base d'un projet de réalisation de programmes. C'est une étape tellement évidente que bien de programmeurs la négligent et prennent en compte des hypothèses implicites qui sont celles des informaticiens et non celles des utilisateurs. Ce genre de choses est à éviter en précisant soigneusement les entrées, les traitements et les sorties du problème.

### 4.2.Conception (analyse)

Il s'agit d'analyser la définition du problème dans ses moindres détails pour une solution satisfaisante. Nous pourrons dire à ce niveau que l'analyse est l'ensemble des tâches de préparation et d'organisation du travail automatisé que devra effectuer l'ordinateur comme la solution spécifique au problème posé.

C'est également au cours de cette étape que les algorithmes sont définis. L'AFNOR (Association Française de Normalisation) définit l'algorithme comme l' « ensemble des règles opératoires et des procédés, définis en vue d'obtenir un résultat déterminé au moyen d'un nombre fini d'opérations ». Nous appuyant sur cette définition, nous dirons qu'il faut lors de l'analyse, sans oublier les opérations de lecture des données et les opérations de restitution des résultats, décomposer et décrire les opérations que l'ordinateur doit effectuer sur les données afin de produire les résultats attendus. Il existe plusieurs méthodes et outils d'analyse.

DESCARTES dans son ouvrage « discours de la méthode » propose de décomposer tout problème en plusieurs problèmes plus petits. Ceux-ci peuvent à leur tour être décomposés jusqu'à aboutir à un ensemble ordonné de problèmes élémentaires pour lesquels la solution est connue.

### 4.3.Codage

Le codage est le terme qui désigne souvent la programmation au sens réduit. C'est l'étape de transcription des algorithmes en ordre exécutables pour l'ordinateur. La transcription se fait au moyen d'un langage de programmation.

### 4.4.Documentation

Comme son nom l'indique, la documentation est l'ensemble des informations nécessaires à la compréhension et à l'utilisation des programmes élaborés. Ainsi la documentation se divise en deux tomes dont le premier s'adresse à des informaticiens programmeurs pour la maintenance des programmes et le second s'adresse aux utilisateurs pour l'utilisation des programmes. Chacun des deux tomes répond à des règles de discipline pour une documentation bonne, complète et précise.

Il est conseillé souvent de commencer la documentation en même temps que le codage et surtout l'actualiser au fur et à mesure des changements. Il faut noter que la documentation sert beaucoup pour les diverses formations relatives aux programmes élaborés.

### 4.5.Tests

C'est une étape au cours de laquelle la validité des programmes est vérifiée par des jeux d'essai. Il se fait des tests informatiques et des tests utilisateurs. Les jeux d'essai doivent comporter aussi bien des cas normaux que des cas d'anomalie et des erreurs volontaires.

En effet, ils ont pour objectifs :

- de tester la validité des contrôles programmés ;
- de contrôler la cohérence des résultats fournis par les programmes en les comparant à ceux obtenus par un autre moyen ;
- de vérifier le comportement des programmes face à des cas particuliers.

Les anomalies constatées donnent lieu à des corrections. C'est une bonne idée, lorsque les moyens le permettent de faire effectuer des tests par un groupe séparé de celui des auteurs programmeurs des programmes ; mais une étroite collaboration des deux groupes donne une meilleure satisfaction (rapidité et précision). Dans tous les cas, un plan de test sérieusement défini au départ doit être suivi.

On distingue trois grandes catégories d'erreurs :

- les erreurs de syntaxe ;
- les erreurs d'exécution ;
- les erreurs de logique.

### 4.6.Mise au point

La mise au point consiste à corriger les erreurs découvertes lors des tests. Tous les programmes, quelle que soit leur taille, contiennent probablement des erreurs (bugs en anglais). Puisque la correction des erreurs ne peut intervenir tant que leur présence n'est pas démontrée, les tests sont indispensables pour localiser les erreurs. Le processus de mise au point est entièrement séparé du processus de test.

La mise au point est habituellement considérée comme l'étape finale de la programmation.

### 4.7.Maintenance

L'étape de maintenance regroupe l'ensemble des travaux nécessaires de retouche ou d'adaptation. Ainsi l'étape de maintenance (séparée de l'étape de programmation) démarre peu de temps après la mise en exploitation des programmes et peut impliquer une évolution.

En effet, malgré la rigueur des tests, l'utilisation d'un programme dans des conditions réelles d'exploitation met toujours en évidence des imperfections telles que :

- l'existence des cas particuliers non encore rencontrés ;
- l'existence d'erreurs de programmation non décelées au cours des tests ;
- les demandes de modification résultantes de l'évolution des règles de gestion ou de la technique.

Les opérations d'actualisation font partie intégrante de l'étape de maintenance qui répond aussi aux principes généraux de conduite de projets informatiques.

# 4. TECHNIQUES DE PROGRAMMATION

Technique = Ensemble des procédés d'un art, d'un métier, etc.

Méthode = Ensemble des procédés raisonnés pour faire une chose.

Un **paradigme de programmation** est un style fondamental de programmation informatique qui traite de la manière dont les solutions aux problèmes doivent être formulées dans un langage de programmation (à comparer à la méthodologie, qui est une manière de résoudre des problèmes spécifiques de génie logiciel).

**4.1 Programmation impérative ou structurée** (C, Pascal)

La programmation structurée constitue un sous-ensemble de la programmation impérative, et un paradigme important de la programmation, apparu vers 1970. Elle dérive de travaux de Nicklaus Wirth pour son Algol W et reçut son coup d'envoi avec l'article fondateur de Dijkstra dans Communications of the ACM nommé *GO TO statement considered harmful*[1].

Elle est en effet célèbre pour son essai de suppression de l'instruction *goto* ou du moins pour la limitation de son usage à des cas inhabituels et graves (que l'on nommerait aujourd'hui (2008) des exceptions).

**4.2 Programmation orientée objet (C++)**

La **programmation orientée objet (POO)** ou **programmation par objet**, est un paradigme de programmation informatique qui consiste en la définition et l'assemblage de briques logicielles appelées *objet* ; un objet représente un concept, une idée ou toute entité du monde physique, comme une voiture, une personne ou encore une page d'un livre.

Orthogonalement à la programmation par objet, afin de faciliter le processus d'élaboration d'un programme, existent des méthodologies de développement logiciel objet dont la plus connue est USDP (Unified Software Development Process).

**4.3 Programmation orientée prototype (Java Script)**

La **programmation orientée prototype** est une forme de programmation orientée objetsans classe, basée sur la notion de prototype. Un **prototype** est un objet à partir duquel on crée de nouveaux objets par clonage. En programmation orientée prototype, les membres d'un objet, attributs et

méthodes, sont appelés **slots** ; il n'y a pas la distinction entre les slots de données et les slots de code qu'on a avec les classes. La grande différence avec la programmation objet à classes est qu'on peut remplacer le contenu des slots, en ajouter d'autres ou changer la hiérarchie d'héritage que cela soit prévu dans l'objet original ou pas.

Self fut le premier langage à prototypes. Il a été conçu dans les laboratoires de Sun dans les années 1990. Le plus connu actuellement est Javascript.

### 4.4 Programmation par contrat (Eiffel)

La **programmation par contrat** est un paradigme de programmation dans lequel le déroulement des traitements est régi par des règles. Ces règles, appelées des assertions, forment un contrat qui précise les responsabilités entre le client et le fournisseur d'un morceau de code logiciel. C'est une méthode de programmation semi-formelle dont le but principal est de réduire le nombre de bogues dans les programmes.

Historiquement, la programmation par contrat a été introduite par Bertrand Meyer dans son langage Eiffel datant de 1985, qui était inspiré de la notation Z créée par Jean-Raymond Abrial.

### 4.5 Programmation déclarative (Latex, HTML)

La **programmation déclarative** est un paradigme de programmation. Il consiste à créer des applications sur la base de composants logiciels indépendant du contexte et ne comportant aucun état interne. Autrement dit, l'appel d'un de ces composants avec les mêmes arguments produit exactement le même résultat, quel que soit le moment et le contexte de l'appel.

En programmation déclarative, on décrit le *quoi*, c'est-à-dire le problème. Par exemple, les pages HTML sont déclaratives car elles décrivent ce que contient une page (texte, titres, paragraphes, etc.) et non comment les afficher (positionnement, couleurs, polices de caractères, etc.). Alors qu'en programmation impérative (par exemple, avec le C ou Java), on décrit le *comment*, c'est-à-dire la solution.

Il existe plusieurs formes de programmation déclarative :

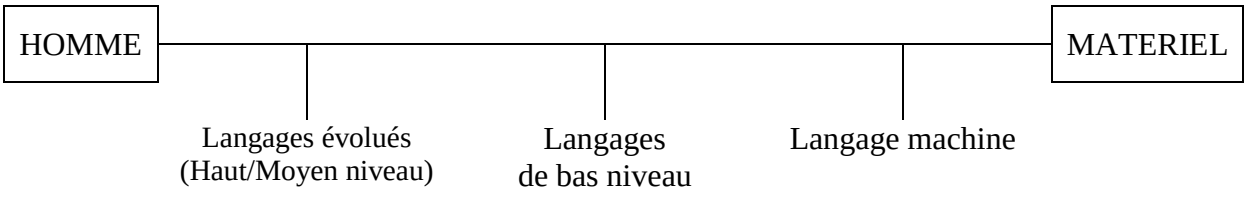
- la programmation descriptive, à l'expressivité réduite, qui permet de décrire des structures de données (par exemple, HTML, XML ou LaTeX),
- la programmation fonctionnelle, qui perçoit les applications comme un ensemble de fonctions mathématiques (LISP, Caml, Haskell),

la programmation logique, pour laquelle les composants d'une application sont des relations logiques (Prolog, Mercury).

# 5. LES ETAPES DE LA PROGRAMMATION

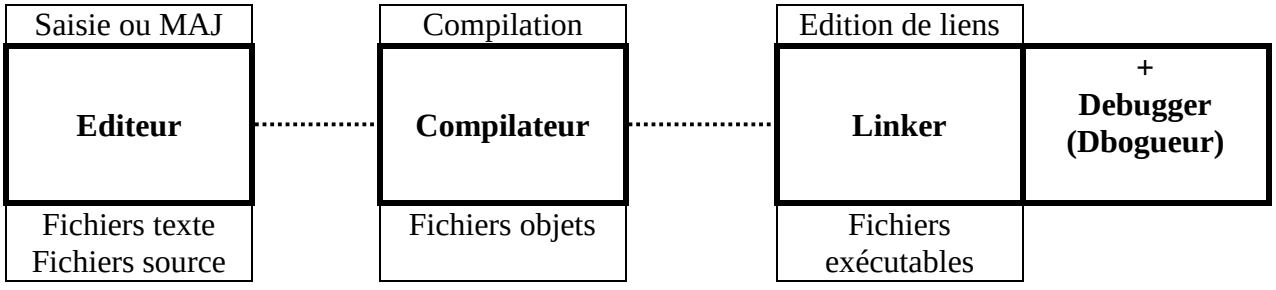
*Etape = Stade = Période ou niveau d'un développement.*

Le langage de l'ordinateur étant le langage binaire (langage machine), un programme écrit en langage binaire se composerait de longues et illisibles séries de 0 et de 1. Une conséquence en est que la détection des erreurs s'avère exclue. On utilise donc, en général, des langages de programmation. Un langage de programmation permet d'écrire des programmes bien lisibles par l'homme et exécutables par le matériel, après une phase de traduction (interprétation ou compilation). Il existe différents langages de programmation que l'on peut classer comme suit :




Ainsi, avant d'être exécuté, un programme source est interprété ou compilé (=> langages interprétés et langages compilés). Tandis qu'un interpréteur traduit et fait exécuter instruction par instruction un programme source, un compilateur traduit un programme source (fichier texte) en un fichier objet (langage machine). Les fichiers objets correspondent généralement à des modules ou ensembles de modules ; aussi les fichiers objets sont ensuite liés pour constituer un fichier exécutable que l'ordinateur peut lire et exécuter directement.

En prenant en considération les langages compilés seuls, voici schématiquement de gauche à droite les transformations que subit un programme :




Un ordinateur ne peut fonctionner que s'il reçoit des ordres de son utilisateur. Ces ordres ou instructions sont utilisés pour constituer des programmes. Dans l'état actuel de la technique, les ordinateurs ne comprennent pas le langage naturel (langage des hommes) ; aussi il a été nécessaire de définir des langages artificiels utilisables pour écrire les programmes d'ordinateur : ce sont des langages de programmation.

Un langage de programmation est donc un ensemble de caractères, de mots, et de règles qui, rigoureusement définis, permettent de communiquer avec un ordinateur sans risque d'ambiguïté. Tout langage de programmation obéit à des règles strictes d'utilisation (orthographe, ponctuation, construction de phrases, etc.).

Le seul langage directement assimilable par l'ordinateur est le langage à base binaire (langage machine). Comme ce langage est très difficile à utiliser par l'homme, on a créé des langages plus commodes dont l'assembleur et les langages évolués universels (ADA, BASIC, COBOL, FORTRAN, APL, PASCAL, PL, etc.).

Pour arriver à un état directement assimilable, un programme en langage évolué subit plusieurs phases comme le présente le schéma ci-dessus.

Ainsi pour maîtriser un langage de programmation, il faut au moins connaître les opérations fondamentales à effectuer à ses différents niveaux de réalisation de programmes, à savoir :

- l'écriture de programmes
- la saisie/mise à jour des codes source et l'enregistrement
- la compilation + correction des erreurs
- l'édition de liens + correction des erreurs
- l'exécution des programmes + correction des erreurs
- l'impression des codes source
- etc.

La saisie et la mise à jour se font habituellement au moyen de l'éditeur de texte pouvant sauvegarder le code source en format ASCII (non formaté ni mis en forme ni mis en page) ; puisque les fichiers formatés comportent beaucoup de codes de contrôle, ces codes de contrôle seraient pris en compte par le compilateur et provoqueraient des erreurs hors programmation. Il est donc vital que les programmes source soient sauvegardés en mode non formaté (ASCII) : l'opération de sauvegarde requiert des notions sur les chemins, les noms et les extensions des fichiers. Notons que bon nombre de langages de programmation comme bon nombre d'applications disposent de leurs propres jeux de caractères.

Pour compiler un programme et faire l'édition de liens, il faut recourir au manuel de référence ou au HELP (aide) du langage de programmation.

Nous essayerons de faire connaître toutes les opérations à effectuer aux différents niveaux de programmation des langages en considérant le cas du langage Pascal. Pascal est généralement utilisé en matière d'enseignement de la programmation pour deux raisons principales : d'abord il est conçu

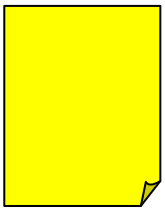
pour l'enseignement ; ensuite il comporte une architecture qui, une fois maîtrisée, permet de se faire la main rapidement dans d'autres langages de programmation.

# 6. NOTIONS DE FICHIER

D'après wikipédia, un fichier est une unité informationnelle physiquement stocké sur un support (mémoire de masse).

### 1. Les caractéristiques d'un fichier

- ✓ Nom
- ✓ Taille
- ✓ Type
- ✓ Emplacement
- ✓ Date de création
- ✓ Date de dernière modification
- ✓ Propriété
- ✓ Droits
- ✓ Contenu



### 2. Deux types de fichiers

- Fichiers programme ;
- Fichiers de données

### 3. Deux natures de fichier

- Fichier binaire ;
- Fichiers text (ASCII)

### 4. Approches de nommage

- Noms courts
- Noms longs

Elle se base sur le travail de ce que l'on fait. On donne le nom en fonction du contenu. On distingue les caractères fichiers et les caractères spécifiques.

Pour les noms courts : 8 caractères pour le nom et 3 caractères pour le format de fichier, les deux sont séparés par un point. (**nom.extension**). Le nom commence entre **A-Z** suivi d'autre(s) lettre(s), et/ou chiffre(s).

# 7. NOTIONS D'INTERFACE ET D'APPLICATION

## 7.1 Interface

L'**interface homme-machine**, **interaction humain-machine** (IHM), **intégration homme-système** (IHS) ou **interface personne-machine** (IPM) définit, les moyens et outils mis en œuvre, afin qu'un humain puisse contrôler et communiquer avec une machine. Les ingénieurs en ce domaine étudient la façon dont les humains interagissent avec les ordinateurs ou entre eux à l'aide d'ordinateurs, ainsi que la façon de concevoir des systèmes qui soient ergonomiques, efficaces, faciles à utiliser ou plus généralement adaptés à leur contexte d'utilisation.

Il existe de nombreuses manières pour un humain et un ordinateur de communiquer. Ces manières sont très dépendantes des dispositifs d'interactions. Ainsi, les premiers ordinateurs étaient utilisés sous forme de traitement par lots : ils étaient alimentés en entrée par des instructions encodées sur des cartes perforées et fournissaient les données de sortie sur des imprimantes. En informatique industrielle, les automates sont encore très souvent pilotés par des baies équipées de boutons poussoirs et de voyants. Un système de pointage tel la souris permettent d'utiliser un ordinateur avec le paradigme WIMP (Windows, Icons, Menus and Pointing device), ce paradigme s'appuie sur les interfaces graphiques pour organiser la présentation d'informations à l'utilisateur. Certaines techniques tentent de rendre l'interaction plus naturelle : la reconnaissance automatique de la parole ou de gestes permettent d'envoyer des informations à un ordinateur ; la synthèse vocale permet d'envoyer un signal audio compréhensible par l'être humain ; les gants de données offre une interaction plus directe que la souris. Les visiocasques essayent d'immerger l'être humain dans une réalité virtuelle, ou d'augmenter la réalité. Les tables interactives permettent un couplage fort entre la manipulation directe par l'être humain sur une surface, et le retour d'information.

D'un point de vue organique, on peut distinguer trois types d'IHM:

**Les interfaces d'acquisition**

- boutons
- molettes, joysticks
- clavier MIDI
- Télécommande
- capteur du mouvement
- microphone avec la reconnaissance vocale.

**Les interfaces de restitution**

- écrans
- LED's témoins
- état visible du système, haut parleur…

**Les interfaces combinées**

- écrans tactiles et Multi-touch
- écrans de type Nano Mod non tactiles
- commandes à retour d'effort
- manette interactive (Wiimote de la wii, Kinect de la XBox, PlayStation Move de la playstation)…

Une **interface en ligne de commande** (couramment abrégé CLI en anglais) est une interface homme-machine dans laquelle la communication entre l'utilisateur et l'ordinateur s'effectue en mode texte :

- l'utilisateur tape du texte au clavier pour demander à l'ordinateur d'effectuer diverses opérations
- l'ordinateur affiche du texte correspondant au résultat de l'exécution des commandes demandées ou à des questions qu'un logiciel pose à l'utilisateur.

Une interface en ligne de commandes peut servir aussi bien pour lancer l'exécution de divers logiciels au moyen d'un interpréteur de commandes, que pour les dialogues avec l'utilisateur de ces logiciels.
L'interface en ligne de commande est la plus ancienne des interfaces conversationnelles développées sur des ordinateurs. Avant cela, les ordinateurs fonctionnaient en traitement par lots : on faisait ingurgiter à l'ordinateur des données enregistrées sur une série de cartes perforées ou une bande perforée. Ces données indiquaient à l'ordinateur quels programmes lancer et de quelles informations ces programmes disposaient pour s'exécuter. Le résultat du traitement (réussi ou erroné) était imprimé sans qu'aucun dialogue avec l'utilisateur ne soit intervenu.
L'apparition des télétypes dans les années 1960, puis, plus tard des consoles à partir des années 1970, qui sont des périphériques qui reçoivent et envoient des caractères à l'ordinateur, a permis le travail sur ordinateur sous la forme de sessions. Le dialogue entre l'utilisateur et l'ordinateur s'effectuait alors en ligne de commandes.
À partir des années 1980, l'apparition de terminaux en mode graphique et des souris a permis les développement des interfaces graphiques, plus appréciés du grand public, peut-être parce qu'elles ne nécessitent pas d'apprendre les noms de différentes commandes avant d'utiliser un ordinateur.
Néanmoins, tout particulièrement sur les systèmes d'exploitations dérivés d'Unix, les interfaces en ligne de commandes restent encore de nos jours appréciées de certains informaticiens, étant donné la richesse de leurs possibilités.
En informatique, le **mode texte**, par opposition au mode graphique, est un type d'affichage sur écran constitué uniquement de caractères.
Le mode texte a pour principal objet l'implémentation d'environnements en mode texte, ce dernier concept d'interface utilisateur étant par ailleurs plus large.
Un écran d'affichage en caractères est généralement composé de 40 ou 80 colonnes, et chaque commande ou action génère un retour à la ligne. Lorsque la ligne atteint la base de l'écran, toutes les lignes sont décalées d'un cran vers le haut et la nouvelle ligne s'inscrit en bas de l'écran. Un écran d'ordinateur compatible PC peut accepter entre 25 et 50 lignes en général - ceci permet un affichage maximal de 80 × 50 = 4 000 caractères à l'écran ; ceux-ci peuvent prendre différentes couleurs (par exemple, 16 couleurs + clignotement) sur différents fonds (16 couleurs). Par contre, le tracé et la chasse (la largeur) des caractères est fixée par le système.
Le mode texte ne permet donc pas l'affichage d'image numérique, sauf par des moyens très détournés comme l'art ASCII. Ce mode était couramment employé sur les ordinateurs compatibles PC durant les années 1980, jusqu'à ce que Windows prenne le pas sur MS-DOS. Les Macintoshs, conçus dès le départ comme ayant une interface graphique, n'ont acquis de terminaux en mode texte qu'avec la sortie de Mac OS X, lequel est désormais un système Unix.
Les programmes en mode texte les plus utilisés sont actuellement les terminaux : des interfaces en texte seulement, ouvertes directement sur la machine hôte, simulées dans une interface graphique.
De par sa souplesse, en matière d'écriture de scripts de commandes notamment, le mode texte reste en 2011 très employé par les utilisateurs des systèmes Unix, notamment, grâce à l'essor de GNU/Linux.
En effet, de par la faible complexité et la petite quantité d'informations transmises dans un sens comme dans l'autre lors de l'envoi de commandes à l'ordinateur et des réponses à celles-ci, la réactivité d'une machine – même pilotée à distance – est sans commune mesure avec celle d'un système de puissance équivalente mais forcé de dédier une partie – souvent importante – de ses ressources à l'affichage de l'interface graphique.

Il est aussi utilisé au démarrage de l'ordinateur par le BIOS pour afficher l'état d'avancement de l'initialisation des périphériques et pour en régler les paramètres.

En informatique, une **interface graphique** (anglais *GUI* pour *graphical user interface*) est un dispositif de dialogue homme-machine, dans lequel les objets à manipuler sont dessinés sous forme de pictogrammes à l'écran, que l'usager peut opérer en imitant la manipulation physique de ces objets avec un dispositif de pointage, le plus souvent une souris. Ce type d'interface a été créé par Xerox en 1981 pour remplacer les interfaces en ligne de commande, puis popularisé par Apple avec l'ordinateur Macintosh, commercialisé en 1984[1],[2].

Les interfaces graphiques sont mises en œuvre par un ensemble de logiciels souvent inclus dans les systèmes d'exploitation. Ce sont des dispositifs courants des appareils informatiques, notamment les ordinateurs, les guichets automatiques bancaires, les téléphones portables et les récepteurs GPS.

### 7.2 Application

En informatique, le terme « **application** » désigne à la fois l'activité d'un utilisateur susceptible d'être automatisée et le logiciel qui automatise cette activité : le « **logiciel applicatif** »[1],[2].

Selon le grand dictionnaire terminologique, un « logiciel applicatif » (ou « logiciel d'application ») est un ensemble de programmes informatiques qui servent à aider un utilisateur à faire un certain travail. Les termes « logiciel applicatif », « application », ou « applicatif » (utilisé comme substantif dans le langage courant) peuvent également être employés.

A **console application** is a computer program designed to be used via a text-only computer interface, such as a text terminal, the command line interface of some operating systems (Unix, DOS, etc.) or the text-based interface included with most Graphical User Interface (GUI) operating systems, such as the Win32 console in Microsoft Windows, the Terminal in Mac OS X, and xterm in Unix. A user typically interacts with a console application using only a keyboard and display screen, as opposed to GUI applications, which normally require the use of a mouse or other pointing device. Many console applications such as command line interpreters are command line tools, but numerous Text User Interface (TUI) programs also exist.
As the speed and ease-of-use of GUI applications have improved over time, the use of console applications has greatly diminished, but not disappeared. Some users simply prefer console based applications, while some organizations still rely on existing console applications to handle key data processing tasks.
The generation of console applications is kept as a feature of modern programming environments such as Visual Studio and the .NET Framework on Microsoft Windows because it greatly simplifies the learning process of a new programming language by removing the complexity of a graphical user interface (see an example in the C Sharp article).
For data processing tasks and computer administration, these programming environments represent the next level of operating system or data processing control after scripting. If an application is only going to be run by the programmer himself and/or a few colleagues, there may be no need for a pretty graphical user interface, leaving the application leaner, faster and easier to maintain.

Une « application de bureau » (« *Desktop application* » en anglais) est un logiciel applicatif qui affiche son interface graphique dans un environnement de bureau[3], il est hébergé et exécuté par l'ordinateur de l'usager. Cette technologie est apparue avec les premiers environnements de bureau en 1970.

Une application Web est un logiciel applicatif qui imite un site web et affiche son interface homme-machine dans un navigateur web. Le logiciel est hébergé par un serveur web transformé. Il est exécuté de facon partagée par les ordinateurs serveur et client (l'usager). Cette technologie est apparue dans les années 1990.