Syntaxe de spécification de format : `printf` et `wprintf` fonctions

https://learn.microsoft.com/fr-fr/cpp/c-runtime-library/format-specification-syntax-printf-and-wprintf-functions?view=msvc-170

- Article
- 03/04/2023
-

Les différentes fonctions `printf` et `wprintf` acceptent une chaîne de format et des arguments facultatifs, et génèrent en sortie une séquence de caractères mise en forme. La chaîne de format contient zéro ou plusieurs *directives* qui sont soit des caractères littéraux pour la sortie, soit des *spécifications de conversion* codées qui décrivent comment mettre en forme un argument dans la sortie. Cet article décrit la syntaxe utilisée pour encoder les spécifications de conversion dans la chaîne de format. Pour obtenir la liste de ces fonctions, consultez E/S de flux.

Une spécification de conversion se compose de champs facultatifs et obligatoires mis en forme comme suit :

%[*flags*][*width*][.*precision*][*size*]*type*

Chaque champ de la spécification de conversion est un caractère ou un nombre qui représente un spécificateur d'option ou de conversion de format particulier. Le champ obligatoire *type* spécifie le genre de conversion à appliquer à un argument. Les champs *d'indicateurs*, *de largeur* et de *précision* facultatifs contrôlent d'autres aspects de format tels que les espaces de début ou les zéros, la justification et la précision affichée. Le champ *size* spécifie la taille de l'argument consommé et converti.

Une spécification de conversion de base contient uniquement le symbole de pourcentage et un caractère *type*. Par exemple, `%s` spécifie une conversion de chaîne. Pour imprimer un caractère de pourcentage, utilisez `%%`. Si un signe de pourcentage est suivi d'un caractère qui n'a aucune signification en tant que champ de format, le gestionnaire de paramètres non valides est appelé. Pour plus d'informations, consultez Validation des paramètres.

Important

Pour la sécurité et la stabilité, assurez-vous que les chaînes de spécification de conversion de format ne sont pas définies par l'utilisateur final. Par exemple, imaginez un programme qui invite l'utilisateur à entrer un nom et enregistre l'entrée dans une variable de chaîne nommée `user_name`. Pour imprimer `user_name`, ne procédez jamais comme suit :

```
printf( user_name ); /* Danger! If user_name contains "%s", program will
crash */
```

Procédez plutôt comme suit :

```
printf( "%s", user_name );
```

Notes

Dans Visual Studio 2015, les `printf` fonctions et de `scanf` la famille de fonctions ont été déclarées en tant que **`inline`** et déplacées vers les `<stdio.h>` en-têtes et `<conio.h>` . Si vous migrez du code plus ancien, vous pouvez voir LNK2019 en relation avec ces fonctions. Pour plus d'informations, consultez Historique des modifications Visual C++ 2003 - 2015.

Spécificateur de conversion de type

Le caractère spécificateur de conversion *type* précise si l'argument correspondant doit être interprété comme un caractère, une chaîne, un pointeur, un entier ou un nombre à virgule flottante. Le caractère *type*, qui est le seul champ de spécification de conversion obligatoire, apparaît après tous les champs facultatifs.

Les arguments qui suivent la chaîne de format sont interprétés en fonction du caractère de *type* correspondant et du préfixe *de taille* facultatif. Les conversions pour les types `char` de caractères et `wchar_t` sont spécifiées à l'aide **`c`** de ou **`C`**, et les chaînes de caractères monooctets et multioctets ou larges sont spécifiées à l'aide **`s`** de ou **`S`**, en fonction de la fonction de mise en forme utilisée. Arguments de caractères et de chaîne qui sont spécifiés à l'aide **`c`** de et **`s`** sont interprétés comme `char` et `char*` par `printf` des fonctions de famille, ou comme `wchar_t` et `wchar_t*` par `wprintf` des fonctions de famille. Arguments de caractères et de chaîne qui sont spécifiés à l'aide **`C`** de et **`S`** sont interprétés comme `wchar_t` et `wchar_t*` par `printf` des fonctions de famille, ou comme `char` et `char*` par `wprintf` des fonctions de famille. Ce comportement est propre à Microsoft. Pour des raisons historiques, les `wprintf` fonctions utilisent **`c`** et **`s`** pour faire référence à `wchar_t` des caractères, et **`C`** spécifient **`S`** des caractères étroits.

Les types entiers tels que `short`, `longint`, `long long`, et leurs `unsigned` variantes sont spécifiés à l'aide **`d`**de , **`i`**, **`o`**, **`u`**, **`x`**et **`X`**. Les types à virgule flottante tels que , et , sont spécifiés à l'aide **`a`**de , **`A`**, **`e`**, **`fE`**, **`F`**, , **`g`**et **`G`**.`long doubledoublefloat` Par défaut, sauf s'ils sont modifiés par un préfixe de *taille* , les arguments entiers sont de type de force et les arguments à `int` virgule flottante sont codés en `double`. Sur les systèmes 64 bits, un `int` est une valeur 32 bits ; par conséquent, les entiers 64 bits sont tronqués lorsqu'ils sont mis en forme pour la sortie, sauf si un préfixe de *taille* de **`ll`** ou **`I64`** est utilisé. Les types de pointeurs spécifiés par utilisent la taille de pointeur par **`p`** défaut pour la plateforme.

Notes

Spécifique à Microsoft :
Le **`z`** caractère de type et le comportement des **`c`**caractères , **`C`**, **`s`**et **`S`** de type lorsqu'ils sont utilisés avec les fonctions et `wprintf` sont des `printf` extensions Microsoft. La norme ISO C utilise **`c`** et **`s`** de manière cohérente pour les caractères et chaînes étroits, et **`cs`** pour les caractères et chaînes larges, dans toutes les fonctions de mise en forme.

Caractères du champ type

Caractère de type	Argument	Format de sortie
c	Caractère	Quand il est utilisé avec les fonctions `printf`, spécifie un caractère codé sur un octet ; quand il est utilisé avec les fonctions `wprintf`, spécifie un caractère large.
C	Caractère	Quand il est utilisé avec les fonctions `printf`, spécifie un caractère large ; quand il est utilisé avec les fonctions `wprintf`, spécifie un caractère codé sur un octet.
d	Integer	Entier décimal signé.
i	Integer	Entier décimal signé.
o	Integer	Entier octal non signé.
u	Integer	Entier décimal non signé.
x	Integer	Entier hexadécimal non signé ; utilise «`abcdef` ».
X	Integer	Entier hexadécimal non signé ; utilise «`ABCDEF` ».
e	Virgule flottante	Valeur signée qui a la forme [-]*ddddd*`e`[|+-]*dd*[*d*], où *d* est un chiffre décimal, *dddd* un ou plusieurs chiffres décimaux en fonction de la précision spécifiée, ou six par défaut, et *dd*[*d*] est deux ou trois chiffres décimaux en fonction du format de sortie et de la taille de l'exposant.
E	Virgule flottante	Identique au **e** format, sauf que plutôt que **eE** d'introduire l'exposant.
f	Virgule flottante	Valeur signée qui a la forme [-]*ddddd*.*dddd*, où *dddd est* un ou plusieurs chiffres décimaux. Le nombre de chiffres avant la virgule décimale dépend de l'ampleur du nombre, et le nombre de chiffres après la virgule décimale dépend de la précision demandée (ou six par défaut).
F	Virgule flottante	Identique au format, sauf que la **f** sortie infini et NaN est capitalisée.
g	Virgule flottante	Les valeurs signées sont affichées au **f** format ou **e** , selon la plus compacte pour la valeur et la précision données. Le **e** format est utilisé uniquement lorsque l'exposant de la valeur est inférieur à -4 ou supérieur ou égal à l'argument *de précision* . Les zéros de droite sont tronqués et la virgule décimale apparaît uniquement si elle est suivie d'un ou plusieurs chiffres.
G	Virgule flottante	Identique au **g** format, sauf que **E**, plutôt que **e**, introduit l'exposant (le cas échéant).
a	Virgule flottante	Valeur hexadécimale à virgule flottante double précision signée qui a la forme [-]`0x`*h.hhhh*[`p`|-+]*dd*, où *h.hhhhh* sont les chiffres hexadécimaux (à l'aide de lettres minuscules) de la mantissa, et *dd* est un ou plusieurs chiffres pour l'exposant. La précision indique le nombre de chiffres après la virgule.
A	Virgule flottante	Valeur hexadécimale à virgule flottante double précision signée qui a la forme [-]`0X`*h.hhhhh*`P`[|-+]*dd*, où *h.hhhhh* sont les chiffres hexadécimaux (à l'aide de lettres

Caractère de type	Argument	Format de sortie
		majuscules) de la mantissa, et *dd* est un ou plusieurs chiffres pour l'exposant. La précision indique le nombre de chiffres après la virgule.
n	Pointeur désignant un entier	Nombre de caractères correctement écrits jusqu'à présent dans le flux ou la mémoire tampon. Cette valeur est stockée dans l'entier dont l'adresse est fournie sous forme d'argument. La taille de l'entier désigné par le pointeur peut être contrôlée par un préfixe de spécification de la taille de l'argument. Le spécificateur **n** est désactivé par défaut ; pour plus d'informations, consultez la remarque importante sur la sécurité.
p	Type de pointeur	Affichez l'argument en tant qu'adresse en chiffres hexadécimaux.
s	String	Quand il est utilisé avec les fonctions `printf`, spécifie une chaîne de caractères codés sur un octet ou multioctets ; quand il est utilisé avec les fonctions `wprintf`, spécifie une chaîne de caractères larges. Les caractères s'affichent jusqu'au premier caractère Null ou jusqu'à ce que la valeur de *precision* soit atteinte.
S	String	Quand il est utilisé avec les fonctions `printf`, spécifie une chaîne de caractères larges ; quand il est utilisé avec les fonctions `wprintf`, spécifie une chaîne de caractères codés sur un octet ou multioctets. Les caractères s'affichent jusqu'au premier caractère Null ou jusqu'à ce que la valeur de *precision* soit atteinte.
Z	Structure `ANSI_STRING` ou `UNICODE_STRING`	**VS 2013 et versions antérieures** Lorsque l'adresse d'une `ANSI_STRING` structure ou `UNICODE_STRING` est passée en tant qu'argument, affichez la chaîne contenue dans la mémoire tampon pointée vers le `Buffer` champ de la structure. Utilisez un préfixe de modificateur de *taille* de **w** pour spécifier un `UNICODE_STRING` argument, par exemple. `%wZ` Le champ `Length` de la structure doit indiquer la longueur, en octets, de la chaîne. Le champ `MaximumLength` de la structure doit indiquer la longueur, en octets, de la mémoire tampon. **Runtime C universel (UCRT)** Il existe un problème connu dans l'UCRT qui est actuellement géré à des fins de compatibilité. Comme le **S** spécificateur, le **Z** spécificateur sans préfixe de modificateur de taille fait référence à un `UNICODE_STRING` lors de l'utilisation d'une fonction d'impression étroite (comme `printf`) et à un lors de l'utilisation d'une fonction d'impression `ANSI_STRING` large (comme `wprintf`). Au lieu de **Z**, utilisez **hZ** pour spécifier un `ANSI_STRING`. **wZ** (ou **lZ**) peut toujours être utilisé pour spécifier un

Caractère de type	Argument	Format de sortie
		`UNICODE_STRING`. En règle générale, le caractère de **`z`** type est utilisé uniquement dans les fonctions de débogage de pilote qui utilisent une spécification de conversion, telles que `dbgPrint` et `kdPrint`.

Dans Visual Studio 2015 et versions ultérieures, si l'argument qui correspond à un spécificateur de conversion à virgule flottante (**`a`**, , **`A`**, **`Efe`**, **`F`**, **`g`**, **`G`**) est infini, indéfini ou NaN, la sortie mise en forme est conforme à la norme C99. Ce tableau répertorie les sorties mises en forme :

Valeur	Output
Infini	`inf`
NaN silencieux	`nan`
NaN signalant	`nan(snan)`
Indefinite NaN	`nan(ind)`

L'une de ces chaînes peut être précédée d'un signe. Si un caractère spécificateur de conversion de *type* virgule flottante est une lettre majuscule, la sortie est également en majuscules. Par exemple, si le spécificateur de format est `%F` et non `%f`, un nombre infini apparaît sous la forme `INF` et non sous la forme `inf`. Les fonctions `scanf` peuvent également analyser ces chaînes. Ces valeurs peuvent donc faire l'aller-retour par l'intermédiaire des fonctions `printf` et `scanf`.

Avant Visual Studio 2015, le CRT utilisait un autre format non standard pour la sortie des valeurs infinies, indéfinies et NaN :

Valeur	Output
+ Infini	`1.#INF`*chiffres aléatoires*
-Infini	`-1.#INF`*chiffres aléatoires*
Indéfini (identique à une valeur NaN silencieuse)	*Chiffre*`.#IND`*chiffres aléatoires*
NaN	*Chiffre*`.#NAN`*chiffres aléatoires*

L'une de ces chaînes peut avoir été précédée d'un signe et avoir été mise en forme différemment en fonction de la largeur et de la précision des champs, parfois avec des effets inhabituels. Par exemple, `printf("%.2f\n", INFINITY)` imprime `1.#J` parce que le *#INF* est « arrondi » à deux chiffres de précision.

Notes

Si l'argument qui correspond à `%s` ou `%S`, ou le champ `Buffer` de l'argument qui correspond à `%Z`, est un pointeur Null, « (Null) » s'affiche.

Notes

Dans tous les formats exponentiels, le nombre par défaut de chiffres d'exposant affichés est de deux (trois si nécessaire). À l'aide de la `_set_output_format` fonction, vous pouvez définir le nombre de chiffres affichés sur trois pour la compatibilité descendante avec le code écrit pour Visual Studio 2013 et avant.

Important

Étant donné que le `%n` format est intrinsèquement non sécurisé, il est désactivé par défaut. Si `%n` est rencontré dans une chaîne de format, le gestionnaire de paramètres non valide est appelé, comme décrit dans Validation des paramètres. Pour activer la `%n` prise en charge, consultez `_set_printf_count_output`.

Directives d'indicateur

Le premier champ facultatif d'une spécification de conversion contient des *directives d'indicateur*. Ce champ contient zéro ou plusieurs caractères d'indicateur qui spécifient la justification de sortie et contrôlent la sortie des signes, des vides, des zéros de début, des virgules décimales et des préfixes octaux et hexadécimaux. Plusieurs directives d'indicateur peuvent apparaître dans une spécification de conversion, et les caractères d'indicateur peuvent apparaître dans n'importe quel ordre.

Caractères d'indicateur

Indicateur	Signification	Default
`-`	Aligner à gauche le résultat selon la largeur de champ donnée.	Aligner à droite.
`+`	Utilisez un signe (+ ou -) pour préfixer la valeur de sortie s'il s'agit d'un type signé.	Le signe apparaît uniquement pour les valeurs signées négatives (-).
`0`	Si *la largeur* est précédée de **`0`**, les zéros de début sont ajoutés jusqu'à ce que la largeur minimale soit atteinte. Si et **`0-`** s'affichent, le **`0`** est ignoré. Si **`0`** est spécifié pour un format entier (**`i`**, , **`ux`**, **`Xo`**, **`d`**) et qu'une spécification de précision est également présente (par exemple, `%04.d`), le **`0`** est ignoré. Si **`0`** est spécifié pour le **`a`** format ou **`A`** à virgule flottante, les zéros de début sont ajoutés au mantissa, après le `0x` préfixe ou `0X` .	Aucun remplissage.
vide (' ')	Utilisez un vide pour préfixer la valeur de sortie si elle est signée et positive. L'espace est ignoré si l'espace et des indicateurs + apparaissent.	Aucun espace ne s'affiche.
`#`	Lorsqu'il est utilisé avec le **`o`**format , **`x`**ou , **`#`** l'indicateur **`x`** utilise `0`, `0x`ou `0X`, pour préfixer une valeur de sortie différente de zéro.	Aucun préfixe n'apparaît.
	Lorsqu'il est utilisé avec le **`e`**format , **`E`**, **`f`**, **`F`**, **`a`**ou , l'indicateur **`A#`** force la valeur de sortie à contenir une virgule décimale.	La virgule décimale apparaît uniquement si des chiffres la suivent.

Indicateur	Signification	Default
	Lorsqu'il est utilisé avec le **`g`** format ou **`G`** , l'indicateur **`#`** force la valeur de sortie à contenir une virgule décimale et empêche la troncation des zéros de fin. Ignoré lorsqu'il est utilisé avec **`c`**, **`d`**, **`i`**, **`u`**ou **`s`**.	La virgule décimale apparaît uniquement si des chiffres la suivent. Les zéros de fin sont tronqués.

Spécification de largeur

Dans une spécification de conversion, le champ facultatif de spécification de largeur apparaît après n'importe quel caractère d'*indicateur*. L'argument *`width`* est un entier décimal non négatif qui contrôle le nombre minimal de caractères qui sont en sortie. Si le nombre de caractères dans la valeur de sortie est inférieur à la longueur spécifiée, des espaces sont ajoutés à gauche ou à droite des valeurs, selon que l'indicateur d'alignement à gauche (`-`) est spécifié ou non, jusqu'à ce que la largeur minimale soit atteinte. Si *`width`* est précédé de 0, les zéros de début sont ajoutés aux conversions d'entiers ou à virgule flottante jusqu'à ce que la largeur minimale soit atteinte, sauf lorsque la conversion est vers un infini ou `NaN`.

La spécification de largeur ne provoque jamais la troncature d'une valeur. Si le nombre de caractères dans la valeur de sortie est supérieur à la largeur spécifiée, ou si *`width`* n'est pas fourni, tous les caractères de la valeur sont de sortie, sous réserve de la spécification de précision.

Si la spécification de la largeur est un astérisque (`*`), un argument `int` issu de la liste d'arguments fournit la valeur. L'argument *`width`* doit précéder la valeur mise en forme dans la liste d'arguments, comme illustré dans cet exemple :

```
printf("%0*d", 5, 3); /* 00003 is output */
```

Une valeur manquante ou petite *`width`* dans une spécification de conversion n'entraîne pas la troncation d'une valeur de sortie. Si le résultat d'une conversion est plus large que la *`width`* valeur, le champ se développe pour contenir le résultat de la conversion.

Spécification de précision

Dans une spécification de conversion, le troisième champ facultatif concerne la spécification de précision. Il se compose d'un point (`.`) suivi d'un entier décimal non négatif qui, selon le type de conversion, spécifie le nombre de caractères de chaîne, le nombre de décimales ou le nombre de chiffres significatifs à générer.

Contrairement à la spécification de largeur, la spécification de précision peut entraîner la troncation de la valeur de sortie ou l'arrondi d'une valeur à virgule flottante. Si *`precision`* est spécifié comme 0 et que la valeur à convertir est 0, le résultat n'est pas de sortie de caractères, comme illustré dans cet exemple :

```
printf( "%.0d", 0 ); /* No characters output */
```

Si la spécification de précision est un astérisque (`*`), un argument `int` issu de la liste d'arguments fournit la valeur. Dans la liste d'arguments, l'argument *`precision`* doit précéder la valeur mise en forme, comme illustré dans cet exemple :

```
printf( "%.*f", 3, 3.14159265 ); /* 3.142 output */
```

Le *`type`* caractère détermine l'interprétation de *`precision`* ou la précision par défaut quand *`precision`* est omis, comme indiqué dans le tableau suivant.

Comment les valeurs de précision affectent le type

Type	Signification	Default
`a`, **`A`**	La précision indique le nombre de chiffres après la virgule.	La précision par défaut s'élève à 13. Si la précision est égale à 0, aucune virgule décimale n'est imprimée, sauf si l'indicateur **`#`** est utilisé.
`c`, **`C`**	La précision n'a aucun effet.	Le caractère est imprimé.
`d`, **`i`**, **`o`**, **`u`**, **`x`**, **`X`**	La précision indique le nombre minimal de chiffres à imprimer. Si le nombre de chiffres dans l'argument est inférieur à *precision*, la valeur de sortie est remplie à gauche de zéros. La valeur n'est pas tronquée lorsque le nombre de chiffres dépasse *la précision*.	La précision par défaut s'élève à 1.
`e`, **`E`**	La précision indique le nombre de chiffres à imprimer après la virgule décimale. Le dernier chiffre imprimé est arrondi.	La précision par défaut s'élève à 6. Si *la précision* est égale à 0 ou si le point (.) s'affiche sans nombre qui la suit, aucune virgule décimale n'est imprimée.
`f`, **`F`**	La valeur de précision indique le nombre de chiffres après la virgule décimale. Si une virgule décimale apparaît, au moins un chiffre apparaît devant. La valeur est arrondie au nombre approprié de chiffres.	La précision par défaut s'élève à 6. Si *la précision* est égale à 0 ou si le point (.) s'affiche sans nombre, aucune virgule décimale n'est imprimée.
`g`, **`G`**	La précision indique le nombre maximal de chiffres significatifs imprimés.	Six chiffres significatifs sont imprimés et tous les zéros de fin sont tronqués.
`s`, **`S`**	La précision indique le nombre maximal de caractères à imprimer. Les caractères au-delà de *precision* ne sont pas imprimés.	Les caractères sont imprimés jusqu'à ce qu'un caractère null soit trouvé.

Spécification de taille d'argument

Dans une spécification de conversion, le champ *size* est un modificateur de longueur d'argument pour le spécificateur de conversion *type*. Le champ *de taille* préfixe au champ de *type* ,**`hhh`** , **`lj`**(minuscules L), **`L`**, **`ll`**, **`wt`**, **`z`**(**`I`** majuscules i), **`I32`**et **`I64`**spécifie la « taille » de l'argument correspondant (long ou court, caractère 32 bits ou 64 bits, un octet ou un caractère large) en fonction du spécificateur de conversion qu'ils modifient. Ces préfixes de taille sont

utilisés avec les caractères de *type* dans les familles `printf` et `wprintf` de fonctions pour spécifier l'interprétation des tailles d'argument, comme illustré dans le tableau suivant. Le champ *size* est facultatif pour certains types d'arguments. Si aucun préfixe de taille n'est spécifié, le formateur consomme les arguments d'entier (par exemple `char`, `short`, `int`, `long` signé ou non signé, ainsi que les types d'énumération) en tant que types `int` 32 bits, tandis que les arguments à virgule flottante `float`, `double` et `long double` sont consommés en tant que types `double` 64 bits. Ce comportement correspond aux règles de promotion d'argument par défaut pour les listes d'arguments de variable. Pour plus d'informations sur la promotion des arguments, consultez Points de suspension et Arguments par défaut dans les expressions Postfix. Sur les systèmes 32 bits et 64 bits, la spécification de conversion d'un argument entier 64 bits doit inclure un préfixe de taille de **`ll`** ou **`I64`**. Sinon, le comportement du formateur n'est pas défini.

Certains types sont de tailles différentes en code 32 bits et 64 bits. Par exemple, la longueur de `size_t` est 32 bits dans le code compilé pour x86, contre 64 bits dans le code compilé pour x64. Pour créer un code de mise en forme indépendant de la plateforme pour les types de largeur variable, vous pouvez utiliser un modificateur de taille d'argument de largeur variable. Au lieu de cela, utilisez un modificateur de taille d'argument 64 bits et promouvez explicitement le type d'argument de largeur variable à 64 bits. Le modificateur de taille de l'argument spécifique **`I`** à Microsoft (majuscules i) gère les arguments entiers de largeur variable, mais nous recommandons les modificateurs spécifiques au **`j`**type , **`t`**et **`z`** pour la portabilité.

Préfixes de taille pour les spécificateurs de type format printf et wprintf

Pour spécifier	Utilisez le préfixe	Avec le spécificateur de type
`char` `unsigned char`	**`hh`**	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`short int` `short unsigned int`	**`h`**	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`__int32` `unsigned __int32`	**`I32`**	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`__int64` `unsigned __int64`	**`I64`**	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`intmax_t` `uintmax_t`	**`j`** ou **`I64`**	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`long double`	**`l`** (minuscules L) ou **`L`**	**`a`**, **`A`**, **`e`**, **`E`**, **`f`**, **`F`**, **`g`** ou **`G`**
`long int` `long unsigned int`	**`l`** (L minuscule)	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`long long int` `unsigned long long int`	**`ll`** (LL minuscules)	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`ptrdiff_t`	**`t`** ou **`I`** (majuscules i)	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
`size_t`	**`z`** ou **`I`** (majuscules i)	**`d`**, **`i`**, **`o`**, **`u`**, **`x`** ou **`X`**
Caractère codé sur un octet	**`h`**	**`c`** ou **`C`**
Caractère large	**`l`** (minuscules L) ou **`w`**	**`c`** ou **`C`**
Chaîne de caractères codés sur un octet	**`h`**	**`s`**, **`S`**ou **`Z`**
Chaîne de caractères larges	**`l`** (minuscules L) ou **`w`**	**`s`**, **`S`**ou **`Z`**

Les types `ptrdiff_t` et `size_t` sont `__int32` et `unsigned __int32` sur les plateformes 32 bits, et `__int64` ou `unsigned __int64` sur les plateformes 64 bits. Les **`I`** préfixes de taille (i majuscules), **`j`**, **`t`**et **`z`** prennent la largeur d'argument correcte pour la plateforme.

Dans Visual C++, bien que `long double` soit un type distinct, il possède la même représentation interne que `double`.

Un **`hc`** spécificateur de type ou **`hC`** est synonyme de **`c`** dans `printf` les fonctions et avec **`C`** dans `wprintf` les fonctions. Un **`lc`**spécificateur de type , **`lCwc`**, ou **`wC`** est synonyme de **`C`** dans `printf` les fonctions et avec **`c`** dans `wprintf` les fonctions. Un **`hs`** spécificateur de type ou **`hS`** est synonyme de **`s`** dans `printf` les fonctions et avec **`S`** dans `wprintf` les fonctions. Un **`ls`**spécificateur de type , **`lS`**, **`ws`**ou **`wS`** est synonyme de **`S`** dans `printf` les fonctions et avec **`s`** dans `wprintf` les fonctions.

Notes

Spécifique à Microsoft :
Les **`I`** préfixes (majuscules i), **`I32`**, **`I64`**et **`w`** des modificateurs de taille d'argument sont des extensions Microsoft et ne sont pas compatibles ISO C. Le **`h`** préfixe lorsqu'il est utilisé avec des données de type `char` et le **`l`** préfixe (L en minuscules) lorsqu'il est utilisé avec des données de type `double` sont des extensions Microsoft.

Voir aussi

`printf`, `_printf_l`, `wprintf`, `_wprintf_l`
`printf_s`, `_printf_s_l`, `wprintf_s`, `_wprintf_s_l`
`printf_p` Paramètres positionnels

Dans cet article