# printf()

La fonction **printf()** combine la capacité de suivi de données, comme la fonction **trace()**avec la capacité de sortir les données et d'autres textes dans le format spécifique décrit. La fonction **printf()** indique à DTrace de suivre les données associées à chaque argument faisant suite au premier argument, puis de formater les résultats en utilisant les règles décrites par le premier argument **printf()** connu sous le nom de *chaîne de format*.

La chaîne de format est une chaîne normale contenant n'importe quel nombre de conversions de format, chacune commençant par le caractère `%` qui décrit comment formater l'argument correspondant. La première conversion dans la chaîne de format correspond au second argument **printf()**, la seconde conversion au troisième argument, etc. L'ensemble du texte entre les conversions est affiché textuellement. Le caractère suivant le caractère de conversion `%` décrit le format à utiliser pour l'argument correspondant.

Contrairement à printf(3C), la fonction **printf()** de DTrace est une fonction intégrée reconnue par le compilateur D. Le compilateur D fournit plusieurs services utiles pour la fonction **printf()** de DTrace, qui sont introuvables dans la bibliothèque C **printf()** :

- Le compilateur D compare les arguments aux conversions en chaîne de format. Si un type d'argument est incompatible avec la conversion de format, le compilateur D fournit un message d'erreur expliquant le problème.
- Le compilateur D ne nécessite pas l'utilisation de préfixes de taille avec les conversions de format **printf. ()** La routine **printf()** C requiert que vous indiquiez la taille des arguments par l'ajout de préfixes tels que `%ld` pour `long` ou `%lld` pour `long long`. Le compilateur D connaît la taille et le type de vos arguments, donc ces préfixes ne sont pas requis dans vos instructions **printf()** D.
- DTrace fournit des caractères de format supplémentaires, utiles pour le débogage et la capacité d'observation. Par exemple, la conversion de format `%a` peut être utilisée pour afficher un pointeur en tant que décalage et nom de symbole.

Afin d'implémenter ces fonctions, la chaîne de format dans la fonction **printf()** de DTrace doit être spécifiée en tant que constante de chaîne dans votre programme D. Les chaînes de format peuvent ne pas être des variables dynamiques de type `string`.

## Spécifications de conversion

Chaque spécification de conversion dans la chaîne de format est introduite par le caractère `%` qui précède les informations suivantes dans la séquence :

- Aucun ou plusieurs *indicateurs* (dans n'importe quel ordre), qui modifient la signification de la spécification de conversion comme décrit dans la section suivante.

- Une *largeur de champ* minimale facultative. Si la valeur convertie dispose d'un nombre d'octets inférieur à la largeur de champ, elle sera complétée par des espaces sur la gauche par défaut, ou sur la droite si l'indicateur d'alignement à gauche (`-`) est spécifié. La largeur de champ peut également être spécifiée par un astérisque (`*`), auquel cas la largeur de champ est définie dynamiquement en fonction de la valeur d'un argument supplémentaire de type `int`.

- Une *précision* facultative, indiquant le nombre minimum de chiffres devant apparaître pour les conversions `d`, `i`, `o`, `u`, `x` et `X` (le champ est complété avec des zéros non significatifs) ; le nombre de chiffres devant apparaître après le caractère de base des conversions `e`, `E` et `f`, le nombre maximum de chiffres significatifs pour les conversions `g` et `G` ou le nombre maximum d'octets à afficher à partir d'une chaîne par la conversion `s`. La précision prend la forme d'un point (`.`) suivi soit d'un astérisque ( `*`), décrit ci-dessous, soit d'une chaîne numérique décimale.

- Une séquence facultative de *préfixes de taille* indiquant la taille de l'argument correspondant, décrite à la section Préfixes de taille . Les préfixes de taille ne sont pas nécessairement en D et sont fournis à des fins de compatibilité avec la fonction **`printf()`** C.

- Un *spécificateur de conversion* indiquant le type de conversion à appliquer à l'argument.

La fonction printf(3C) prend également en charge des spécifications de conversion sous le format `%` *n* `$` où *n* est en nombre entier décimal ; la fonction **`printf()`** de DTrace ne prend pas en charge ce type de spécification de conversion.

# Spécificateurs d'indicateurs

Les indicateurs de conversion **`printf()`** sont activés en spécifiant au moins l'un des caractères suivants, susceptibles d'apparaître dans n'importe quel ordre :

`'`

La partie de nombre entier du résultat d'une conversion décimale (`%i`, `%d`, `%u`, `%f`, `%g` ou `%G`) est formatée avec des milliers de caractères de groupement utilisant le caractère de groupement non monétaire. Certains environnements linguistiques, notamment l'environnement linguistique POSIX C ne fournissent pas de caractères de groupement non monétaires utilisés avec cet indicateur.

`-`

Le résultat de la conversion est justifié à gauche dans le champ. La conversion est justifiée à droite si cet indicateur n'est pas spécifié.

`+`

Le résultat d'une conversion signée commence toujours par un signe (+ ou -). Si cet indicateur n'est pas spécifié, la conversion commence par un signe uniquement lorsqu'une valeur négative est convertie.

`space`

Si le premier caractère d'une conversion signée n'est pas un signe ou si une conversion signée ne renvoie aucun caractère, un espace est placé avant le résultat. Si les indicateurs `space` et + sont tous les deux affichés, l'indicateur d'espace est ignoré.

`#`

La valeur est convertie en un format secondaire s'il y en a un de défini pour la conversion sélectionnée. Les formats secondaires pour des conversions sont décrits avec la conversion correspondante.

0

Pour les conversions `d`, `i`, `o`, `u`, `x`, `X`, `e`, `E`, `f`, `g` et `G`, des zéros non significatifs (suivis par n'importe quelle indication de signe ou de base) sont utilisés pour remplir la largeur de champ. Aucun remplissage avec des espaces n'est réalisé. Si les indicateurs `0` et `-` sont tous les deux affichés, l'indicateur `0` est ignoré. Pour les conversions `d`, `i`, `o`, `u`, `x` et `X`, si une précision est spécifiée, l'indicateur `0` est ignoré. Si les indicateurs `0` et `'` sont tous les deux affichés, les caractères de groupement sont insérés avant le remplissage de zéros.

# Spécificateurs de largeur et de précision

La largeur de champ minimum peut être spécifiée en tant que chaîne numérique décimale précédée de n'importe quel spécificateur d'indicateur, auquel cas la largeur de champ est définie sur le nombre spécifié de colonnes. La largeur de champ peut également être spécifiée en tant qu'astérisque (`*`) auquel cas on accède à un argument supplémentaire de type `int` pour déterminer la largeur de champ. Par exemple, pour afficher un nombre entier `x` dans une largeur de champ déterminée par la valeur de la variable `int w`, vous devez écrire l'instruction D suivante :

```
printf("%*d", w, x);
```

La largeur de champ peut également être spécifiée en utilisant un caractère `?` pour indiquer que la largeur de champ doit être définie en fonction du nombre de caractères requis pour formater une adresse en hexadécimal dans le modèle de données du noyau du système d'exploitation. La largeur est définie sur 8 si le noyau utilise un modèle de données à 32 bits ou sur 16 s'il en utilise un à 64 bits.

La précision pour la conversion peut être spécifiée en tant que chaîne numérique décimale suivie d'un point (`.`) ou d'un astérisque précédé d'un ( `*`) point. Si un astérisque est utilisé pour spécifier la précision, on accède à un argument supplémentaire de type `int` avant la conversion pour déterminer la précision. Si la largeur et la précision sont précisées en tant qu'astérisques, l'ordre des arguments de **`printf()`** pour la conversion doit apparaître dans l'ordre suivant. largeur, précision, valeur.

# Préfixes de taille

Les préfixes de taille sont requis dans les programmes ANSI-C utilisant printf(3C) afin d'indiquer la taille et le type de l'argument de conversion. Le compilateur D exécute ce traitement pour vos appels **`printf()`**, donc les préfixes de taille ne sont pas requis. Bien que les préfixes de taille soient fournis à des fins de compatibilité avec C, leur utilisation est fortement déconseillée dans les programmes D car ils établissent une liaison entre votre code et un modèle de données particulier lors de l'utilisation de types dérivés. Par exemple, si un `typedef` est redéfini sur des types de base de nombres entiers différents sur le modèle de données, il n'est pas possible d'utiliser une conversion C unique fonctionnant sur les deux modèles de données sans connaître de manière explicite les deux types sous-jacents et inclure une expression de forçage de type, ou définir plusieurs chaînes de format. Le compilateur D résout ce problème automatiquement en vous autorisant à omettre les préfixes de taille et en déterminant automatiquement la taille de l'argument.

Les préfixes de taille peuvent précéder le nom de la conversion de format et être placés après tout spécificateur d'indicateur, de largeur et de précision. Les préfixes de taille se présentent comme suit :

- Un `h` facultatif spécifie qu'une conversion suivante `d`, `i`, `o`, `u`, `x` ou `X` s'applique à un `short` ou à un `unsigned short`.
- Un `l` facultatif spécifie qu'une conversion suivante `d`, `i`, `o`, `u`, `x` ou `X` s'applique à un `long` ou à un `unsigned long`.

- Un `ll` facultatif spécifie qu'une conversion suivante `d`, `i`, `o`, `u`, `x` ou `X` s'applique à un `long long` ou à un `unsigned long long`.
- Un `L` facultatif spécifie qu'une conversion suivante `e`, `E`, `f`, `g` ou `G` s'applique à un `long double`.
- Un `l` facultatif spécifie qu'une conversion suivante `c` s'applique à un argument `wint_t` et qu'une conversion suivante `s` s'applique à un pointeur vers un argument `wchar_t`.

# Formats de conversion

Chaque séquence de caractères de conversion provoque l'extraction d'aucun ou de plusieurs arguments. Si un nombre insuffisant d'arguments est fourni pour la chaîne de format, ou si la chaîne de format est épuisée et que des arguments restent disponibles, le compilateur D génère un message d'erreur approprié. Si un format de conversion est spécifié, le compilateur D génère un message d'erreur approprié. Les séquences de caractères de conversion sont les suivantes :

`a`

Le pointeur ou l'argument `uintptr_t` est affiché en tant que nom du symbole de noyau sous le format *module* `` `*nom-symbole* `` plus un décalage d'octets hexadécimaux facultatif. Si la valeur ne se situe pas dans la plage définie par un symbole de noyau connu, elle est affichée en tant que nombre entier hexadécimal.

`c`

L'argument `char`, `short` ou `int` est affiché en tant que caractère ASCII.

`C`

L'argument `char`, `short` ou `int` est affiché en tant que caractère ASCII si le caractère est un caractère ASCII imprimable. Dans le cas contraire, il est affiché avec la séquence d'échappement correspondante, comme illustré dans le Tableau 2–5.

`d`

L'argument `char`, `short`, `int`, `long` ou `long long` est affiché en tant que nombre entier décimal (base 10). Si l'argument est `signed`, il sera affiché en tant que valeur signée. Si l'argument est `unsigned`, il sera affiché en tant que valeur non signée. Cette conversion a la même signification que `i`.

`e, E`

L'argument `float`, `double` ou `long double` est converti sur le style `[-]` *d*`.`*ddd*`e±` *dd*, le caractère de base étant précédé d'un chiffre et suivi d'un nombre de chiffres égal à la précision. Le caractère de base n'est pas égal à zéro si l'argument n'est pas égal à zéro. Si aucune précision n'est spécifiée, sa valeur par défaut est 6. Si la précision est égale à 0 et l'indicateur # n'est pas spécifié, aucun caractère de base n'apparaît. Le format de conversion `E` produit un nombre avec `E` au lieu de `e` introduisant l'exposant. L'exposant contient toujours au moins deux chiffres. La valeur est arrondie au nombre approprié de chiffres.

`f`

L'argument `float`, `double` ou `long double` est converti sur le style `[-]` *ddd*`.` *ddd*, le nombre de chiffres précédant le caractère de base étant égal à la spécification de précision. Si aucune précision n'est spécifiée, sa valeur par défaut est 6. Si la précision est égale à 0 et l'indicateur # n'est pas spécifié, aucun caractère de base n'apparaît. Si un caractère de base s'affiche, il est précédé d'au moins un chiffre. La valeur est arrondie au nombre approprié de chiffres.

`g, G`

L'argument `float`, `double` ou `long double` est affiché dans le style `f` ou `e` (ou dans le style `E` dans le cas d'un caractère de conversion `G`), avec la précision spécifiant le nombre de chiffres significatifs. Si une précision explicite est égale à 0, elle prend 1 comme valeur. Le style utilisé dépend de la valeur convertie : le style `e` (ou `E`) n'est utilisé que si l'exposant résultant de la conversion est inférieur à -4 ou est supérieur ou égal à la précision. Les zéros finaux sont supprimés de la partie fractionnelle du résultat. Un caractère de base ne s'affiche que s'il précède un chiffre. Si l'indicateur # est spécifié, les zéros finaux ne sont pas supprimés du résultat.

`i`

L'argument `char`, `short`, `int`, `long` ou `long long` est affiché en tant que nombre entier décimal (base 10). Si l'argument est `signed`, il sera affiché en tant que valeur signée. Si l'argument est `unsigned`, il sera affiché en tant que valeur non signée. Cette conversion a la même signification que `d`.

`o`

L'argument `char`, `short`, `int`, `long` ou `long long` est affiché en tant que nombre entier octal non signé (base 8). Les arguments `signed` ou `unsigned` peuvent être utilisés avec cette conversion. Si l'indicateur # est spécifié, la précision du résultat sera augmentée si nécessaire pour forcer le premier chiffre du résultat à prendre la valeur zéro.

`p`

Le pointeur ou l'argument `uintptr_t` est affiché en tant que nombre entier hexadécimal (base 16). D accepte des arguments de pointeur de n'importe quel type. Si l'indicateur # est spécifié, `0x` sera pré-ajouté aux résultats différents de zéro.

`s`

L'argument doit être un tableau de `char` ou une `string`. Les octets de ce tableau ou de la `string` sont lus jusqu'à un caractère nul de fin ou jusqu'à la fin des données, puis interprétés et affichés en tant que caractères ASCII. Si la précision n'est pas spécifiée, elle prend une valeur infinie, donc tous les caractères jusqu'au premier caractère nul sont affichés. Dans le cas contraire, seule la partie du tableau de caractères affichée dans le nombre correspondant de colonnes à l'écran s'affiche. Si un argument de type `char *` doit être formaté, son type doit être forcé sur `string` ou l'opérateur D `stringof` doit être placé en préfixe pour indiquer que DTrace doit procéder au suivi des octets de la chaîne et les formater.

`S`

L'argument doit être un tableau de `char` ou une `string`. L'argument est traité comme avec la conversion `%s` mais tous les caractères ASCII non imprimables sont remplacés par la séquence d'échappement correspondante décrite dans le Tableau 2–5.

`u`

L'argument `char`, `short`, `int`, `long` ou `long long` est affiché en tant que nombre entier décimal non signé (base 10). Les arguments `signed` ou `unsigned` peuvent être utilisés avec cette conversion et le résultat est toujours formaté en tant que `unsigned`.

`wc`

L'argument `int` est converti en caractère large (`wchar_t`) et celui qui en résulte s'affiche.

`ws`

L'argument doit être un tableau de `wchar_t`. Les octets d'un tableau sont lus jusqu'à un caractère nul de fin ou la fin des données, puis sont interprétés et affichés en tant que caractères larges. Si la précision n'est pas spécifiée, elle prend une valeur infinie, donc tous les caractères larges jusqu'au premier caractère nul sont affichés. Dans le cas contraire, seule la partie du tableau de caractères larges affichée dans le nombre correspondant de colonnes à l'écran s'affiche.

`x, X`

L'argument `char`, `short`, `int`, `long` ou `long long` est affiché en tant que nombre entier décimal non signé (base 16). Les arguments `signed` ou `unsigned` peuvent être utilisés avec cette conversion. Si le format de conversion `x` est utilisé, les chiffres littéraux `abcdef` sont utilisés. Si le format de conversion `X` est utilisé, les chiffres littéraux `abcdef` sont utilisés. Si l'indicateur `#` est spécifié, `0x` (pour `%x`) ou `0X` (pour `%X`) y sera ajouté.

`Y`

L'argument `uint64_t` est interprété comme le nombre de nanosecondes écoulées depuis 00:00 Temps universel coordonné, 1er janvier 1970 et s'affiche sous le format cftime(3C) suivant : "`%Y %a %b %e %T %Z`". Le nombre actuel de nanosecondes écoulées depuis le 1er janvier 1970, 00:00 UTC, est disponible dans la variable `walltimestamp`.

`%`

Permet d'afficher un caractère `%` littéral. Aucun argument n'est converti. La spécification de conversion complète doit être `%%`.


- *Previous*: Chapitre 11 Tampons et mise en tampon
- *Next*: printa()