Langage C++ - Les types de données

Septembre 2015

Les types de données

Les données manipulées en langage C++, comme en langage C, sont typées, c'est-à-dire que pour chaque donnée que l'on utilise (dans les variables par exemple) il faut préciser le type de donnée, ce qui permet de connaître l'occupation mémoire (le nombre d'octets) de la donnée ainsi que sa représentation :

- **des nombres** : entiers (**int**) ou **réels**, c'est-à-dire à virgules (**float**)
- **des pointeurs** (en anglais *pointers*) : permettent de stocker l'adresse d'une autre donnée, ils « pointent » vers une autre donnée

En C++ il existe plusieurs types d'entiers, dépendant du nombre d'octets sur lesquels ils sont codés ainsi que de leur format, c'est-à-dire s'ils sont signés (possédant le signe - ou +) ou non. Par défaut les données sont signées.

De plus, le langage C++ introduit par rapport au langage C un nouveau type de donnée appelé *bool*. Ce type de variable accepte deux états :

- **true** (vrai) : correspondant à la valeur 1
- **false** (faux) : correspondant à la valeur 0

Etant donné que ce type de donnée est codé sur 8 bits (un octet), 7 de ces bits ne servent à rien, puisqu'une variable de ce type ne peut prendre que deux états. En réalité, toutes les valeurs différentes de zéro sont considérées comme vraies (donc considérées comme égales à 1).

Voici un tableau donnant les types de données en langage C++ :

Type de donnée	Signification	Taille (en octets)	Plage de valeurs acceptée
char	Caractère	1	-128 à 127
unsigned char	Caractère non signé	1	0 à 255
short int	Entier court	2	-32 768 à 32 767
unsigned short int	Entier court non signé	2	0 à 65 535
int	Entier	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	-32 768 à 32 767 -2 147 483 648 à 2 147 483 647

unsigned int	Entier non signé	2 (sur processeur 16 bits) 4 (sur processeur 32 bits)	0 à 65 535 0 à 4 294 967 295
long int	Entier long	4	-2 147 483 648 à 2 147 483 647
unsigned long int	Entier long non signé	4	0 à 4 294 967 295
float	Flottant (réel)	4	$-3.4*10^{-38}$ à $3.4*10^{38}$
double	Flottant double	8	$-1.7*10^{-308}$ à $1.7*10^{308}$
long double	Flottant double long	10	$-3.4*10^{-4932}$ à $3.4*10^{4932}$
bool	Booléen	Même taille que le type *int*, parfois *1* sur quelques compilateurs	Prend deux valeurs : *'true'* et *'false'* mais une conversion implicite (valant 0 ou 1) est faite par le compilateur lorsque l'on affecte un entier (en réalité toute autre valeur que 0 est considérée comme égale à *True*).

Nombre entier (*int*)

Un nombre entier est un nombre sans virgule qui peut être exprimé dans différentes bases :

- Base décimale : l'entier est représenté par une suite de chiffres unitaires (de 0 à 9) ne devant pas commencer par le chiffre 0
- Base hexadécimale : l'entier est représenté par une suite d'unités (de 0 à 9 ou de A à F (ou a à f)) devant commencer par *0x* ou *0X*
- Base octale : l'entier est représenté par une suite d'unités (incluant uniquement des chiffres de 0 à 7) devant commencer par *0*

Les entiers sont signés par défaut, cela signifie qu'ils comportent un signe. Pour stocker l'information concernant le signe (en binaire), les ordinateurs utilisent le complément à deux.

Nombre à virgule (*float*)

Un nombre à virgule flottante est un nombre à virgule, il peut toutefois être représenté de différentes façons :

- un entier décimal : 895
- un nombre comportant un point (et non une virgule) : 845.32
- une fraction : 27/11
- un nombre exponentiel, c'est-à-dire un nombre (éventuellement à virgule) suivi de la lettre *e* (ou *E*), puis

d'un entier correspondant à la puissance de 10 (signé ou non, c'est-à-dire précédé d'un + ou d'un -) : 2.75e-2 35.8E+10 .25e-2

En réalité, les nombres réels sont des nombres à virgule flottante, c'est-à-dire un nombre dans lequel la position de la virgule n'est pas fixe, et est repérée par une partie de ses bits (appelée l'exposant), le reste des bits permettent de coder le nombre sans virgule (la mantisse).

Les nombres de type **float** sont codés sur 32 bits dont :

- 23 bits pour la mantisse
- 8 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **double** sont codés sur 64 bits dont :

- 52 bits pour la mantisse
- 11 bits pour l'exposant
- 1 bit pour le signe

Les nombres de type **long double** sont codés sur 80 bits dont :

- 64 bits pour la mantisse
- 15 bits pour l'exposant
- 1 bit pour le signe

La précision des nombres réels est approchée. Elle dépend du nombre de positions décimales, suivant le type de réel elle sera au moins :

- de 6 chiffres après la virgule pour le type **float**
- de 15 chiffres après la virgule pour le type **double**
- de 17 chiffres après la virgule pour le type **long double**

Caractère (*char*)

Le type **char** (provenant de l'anglais *character*) permet de stocker la valeur ASCII d'un caractère, c'est-à-dire un nombre entier !

Par défaut les nombres sont signés, cela signifie qu'ils comportent un signe. Pour stocker l'information concernant le signe (en binaire), les ordinateurs utilisent le complément à deux. Une donnée de type char est donc signée, cela ne signifie bien sûr pas que la lettre possède un signe mais tout simplement que dans la mémoire la valeur codant le caractère peut être négative...

Si jamais on désire par exemple stocker la lettre B (son code ASCII est 66), on pourra définir cette donnée soit par le nombre 66, soit en notant 'B' où les apostrophes simples signifient *code ascii de...*

Il n'existe pas de type de données pour les chaînes de caractères (suite de caractères) en langage C. Pour créer une chaîne de caractères on utilisera donc des tableaux contenant dans chacune de ses cases un caractère...

Créer un type de donnée

Il est possible en C++ comme en C de définir un nouveau type de données grâce au mot clé *typedef*. Celui-ci admet la syntaxe suivante :

```
typedef Caracteristiques_du_type Nom_du_type
```

où

- *Caracteristiques_du_type* représente un type de données existant (par exemple *float*, *short int*, ...)
- *Nom_du_type* définit le nom que vous donnez au nouveau type de donnée

Ainsi l'instruction suivante crée un type de donnée *Ch* calqué sur le type *char* :

```
typedef char Ch
```

Conversion de type de données

On appelle *conversion de type de données* le fait de modifier le type d'une donnée en une autre. Il peut arriver par exemple que l'on veuille travailler sur un type de variable, puis l'utiliser sous un autre type. Imaginons que l'on travaille par exemple sur une variable en virgule flottante (type *float*), il se peut que l'on veuille « supprimer les chiffres après la virgule », c'est-à-dire convertir un *float* en *int*. Cette opération peut être réalisée de deux manières :

- **conversion implicite** : une conversion implicite consiste en une modification du type de donnée effectuée automatiquement par le compilateur. Cela signifie que lorsque l'on va stocker un type de donnée dans une variable déclarée avec un autre type, le compilateur ne retournera pas d'erreur mais effectuera une conversion *implicite* de la donnée avant de l'affecter à la variable.

```
int x;

x = 8.324;
```

x contiendra après affectation la valeur 8.

- **conversion explicite** : une conversion explicite (appelée aussi *opération de cast*) consiste en une modification du type de donnée forcée. Cela signifie que l'on utilise un opérateur dit *de cast* pour spécifier la conversion. L'opérateur de cast est tout simplement le type de donnée, dans lequel on désire convertir une variable, entre des parenthèses précédant la variable.

```
int x;

x = (int)8.324;
```

x contiendra après affectation la valeur 8.
De plus, le langage C++ rajoute une notation fonctionnelle pour faire une conversion explicite, la syntaxe de cette notation est :

```
int x;

x = int(8.324);
```

Le texte original de cette fiche pratique est extrait de «Tout sur le C++» (Christine EBERHARDT, Collection CommentCaMarche.net, Dunod, 2009)

‹ Précédent

- 1
- 2
- 3

Réalisé sous la direction de Jean-François PILLOU,
fondateur de CommentCaMarche.net.