Introduction au langage C

5 janvier 2010

Table des matières

Chapitre 1

Variables, types, et déclarations

1.1 Les données dans un programme

Un programme manipule des données de différents types sous la forme de constantes ou de variables.

1.1.1 Notion de type

La mémoire centrale est un ensemble de *positions binaires* nommées bits. Les bits sont regroupés en octets (8 bits), et chaque octet est repéré par une adresse. L'ordinateur, compte tenu de sa technologie (actuelle !), ne sait représenter et traiter que des informations exprimées sous forme binaire. Toute information, quelle que soit sa nature, devra être codée sous cette forme. Dans ces conditions, on voit qu'il ne suffit pas de connaître le contenu d'un emplacement de la mémoire (d'un ou de plusieurs octets) pour être en mesure de lui attribuer une signification. Par exemple, si *vous* savez qu'un octet contient le *motif binaire* suivant : 01001101 vous pouvez considérer que cela représente le nombre entier 77 (puisque le motif ci-dessus correspond à la représentation en base 2 de ce nombre). Mais pourquoi cela représenterait-il un nombre ? En effet, toutes les informations (nombres entiers, nombres réels, nombres complexes, caractères, instructions de programme en langage machine, graphiques...) devront, au bout du compte, être codées en binaire. Dans ces conditions, les huit bits ci-dessus peuvent peut-être représenter un caractère ; dans ce cas, si nous connaissons la convention employée sur la machine concernée pour représenter les caractères, nous pouvons lui faire correspondre un caractère donné (par exemple M, dans le cas du code ASCII). On comprend donc qu'il n'est pas possible d'attribuer une signification à une information binaire tant que l'on ne connaît pas la manière dont elle a été codée. Qui plus est, en général, il ne sera même pas possible de *traiter* cette information. Par exemple, pour additionner deux informations, il faudra savoir quel codage a été employé afin de pouvoir mettre en œuvre les *bonnes* instructions (en langage machine). D'une manière générale, la notion de type, telle qu'elle existe dans les langages évolués, sert à régler (entre autres choses) les problèmes que nous venons d'évoquer.

Les types de base du langage C se répartissent en trois grandes catégories en fonction de la nature des informations qu'ils permettent de représenter :

– nombres entiers (mot clé **int**) ;
– nombres réels simple précision (mot clé **float**) ;
– nombres réels double précision (mot clé **double**) ;
– caractères (mot clé **char**) ; nous verrons qu'en fait char apparaît (en C) comme un cas particulier de int.

Ces types peuvent par ailleurs être qualifiés par les qualificateurs suivants :

– **short** : format plus court que le type de base ;
– **long** : format plus long que le type de base ;
– **signed** : nombre signé ;
– **unsigned** : nombre non signé ;

Ces qualificateurs s'appliquent essentiellement au nombres entiers. Par exemple `unsigned short int` est le type d'un entier non signé codé sur 16 bits. Il existe aussi un type particulier : **void** qui représente en fait une absence de type.

1.1.2 Le type entier

Le mot clé **int** correspond à la représentation de nombres entiers relatifs. Pour ce faire : un bit est réservé pour représenter le signe du nombre (en général 0 correspond à un nombre positif) ; les autres bits servent à représenter la valeur absolue du nombre.

Les différents types d'entiers

Le C prévoit que, sur une machine donnée, on puisse trouver jusqu'à trois *tailles* différentes d'entiers, désignées par les mots clés suivants :
– short
– int (c'est celui que nous utiliserons systématiquement),
– long int.
Chaque taille impose naturellement ses limites. Toutefois, ces dernières dépendent, non seulement du mot clé considéré, mais également de la machine utilisée : tous les int n'ont pas la même taille sur toutes les machines ! Fréquemment, deux des trois mots clés correspondent à une même taille (par exemple, sur PC, **short** et **int** correspondent à 16 bits, tandis que **long** correspond à 32 bits). A titre indicatif, avec 16 bits, on représente des entiers s'étendant de -32 768 à 32 767 ; avec 32 bits, on peut couvrir les valeurs allant de -2 147 483 648 à 2 147 483 647.

Remarque : en toute rigueur, chacun des trois types (short, int et long) peut être nuancé par l'utilisation du *qualificatif* **unsigned** (non signé). Dans ce cas, il n'y a plus de bit réservé au signe et on ne représente plus que des nombres positifs. Son emploi est réservé à des situations particulières.

1.1.3 Les types flottants

Les différents types flottants et leur représentation en mémoire

Les types **flottants** permettent de représenter, de manière approchée, une partie des nombres réels. Pour ce faire, ils s'inspirent de la notation *scientifique* (ou *exponentielle*) bien connue qui consiste à écrire un nombre sous la forme 1.5.1022 ou 0.472 .10-8 ; dans une telle notation, on nomme *mantisses* les quantités telles que 1.5 ou 0.472 et *exposants* les quantités telles que 22 ou -8.

Le C prévoit trois types de flottants correspondant à des tailles différentes : **float**, **double** (c'est celui que l'on emploie systématiquement) et **long double**. La connaissance des caractéristiques exactes du système de codage n'est généralement pas indispensable. En revanche, il est important de noter que de telles représentations sont caractérisées par deux éléments :

- la précision : lors du codage d'un nombre décimal quelconque dans un type flottant, il est nécessaire de ne conserver qu'un nombre fini de bits. Or la plupart des nombres s'exprimant avec un nombre limité de décimales ne peuvent pas s'exprimer de façon exacte dans un tel codage. On est donc obligé de se limiter à une représentation approchée en faisant ce qu'on nomme une "erreur de troncature". Quelle que soit la machine utilisée, on est assuré que cette erreur (relative) ne dépassera pas 10-6 pour le type float et 10-10 pour le type long double ;
- le domaine couvert, c'est-à-dire l'ensemble des nombres représentables à l'erreur de troncature près. Là encore, quelle que soit la machine utilisée, on est assuré qu'il s'étendra au moins de 10-37 à 10+37.

Notation des constantes flottantes

Par défaut, toutes les constantes sont créées par le compilateur dans le type double.

1.1.4 Les types caractères

1.1.4.1 La notion de caractère en langage C

Le C permet, comme le Pascal, de manipuler des caractères codés en mémoire sur un octet. Bien entendu, le code employé, ainsi que l'ensemble des caractères représentables, dépend de l'environnement de programmation utilisé (c'est-à-dire à la fois de la machine concernée et du compilateur employé). Néanmoins, on est toujours certain de disposer des lettres (majuscules et minuscules), des chiffres, des signes de ponctuation et des différents séparateurs (en fait, tous ceux que l'on emploie pour écrire un programme !). En revanche, les caractères "nationaux" (caractères accentués ou ç) ou les caractères "semi-graphiques" ne se rencontrent pas dans tous les environnements. Par ailleurs, la notion de caractère en C dépasse celle de caractère imprimable, c'est-à-dire auquel est obligatoirement associé un graphisme (et qu'on peut donc imprimer ou afficher sur un écran). C'est ainsi qu'il existe certains "ca-

ractères" de changement de ligne, de tabulation, d'activation d'une alarme sonore (cloche),... Nous avons d'ailleurs déjà utilisé le premier (sous la forme
n).

1.1.4.2 Notation des constantes caractères

Les constantes de type *caractère*, lorsqu'elles correspondent à des caractères imprimables, se notent de façon classique, en écrivant entre apostrophes (ou **quotes)** le caractère voulu, comme dans ces exemples :'Y', '+', '$'. ; Certains caractères non imprimables possèdent une représentation conventionnelle utilisant le caractère
) ; il s'agit des caractères (', " et ?) qui, bien que disposant d'un graphisme, jouent un rôle particulier de délimiteur qui les empêche d'être notés de manière classique entre deux apostrophes.

Voici la liste de ces caractères.

Notation usuelle en code ASCII (abréviation EN C) (hexadécimal)

```
\a 07 BEL cloche ou bip (alert ou audible bell)
\b 08 BS Retour arrière (Backspace)
\f 0C FF Saut de page (Form Feed)
\n 0A LF Saut de Ligne (Line Feed)
\r 0D CR Retour chariot (Carriage Return)
\t 09 HT Tabulation horizontaLe (HorizontaL Tab)
\v 0B VT Tabutation verticale (VerticaL Tab)
\\ SC \
\' 2C '
\ '' 22 "
\? 3F ?
```

1.2 Les Variables

Les données d'un programme sont stockées en mémoire et sont manipulées par l'intermédiaire d'identificateurs de variables. Par exemple la déclaration : `int n` réserve de l'espace en mémoire pour un entier dont l'identificateur (nom qui permet de le manipuler) est `n`. En C, comme en UNIX, on distingue les majuscules des minuscules. Ainsi la déclaration suivantes `int n` est différente de `int N`.

1.3 Les Constantes

Une constante entière peut s'écrire dans les systèmes décimal et hexadécimal.

Exemples : 12, 0x35. Par défaut elle est de type int.

Une constante réelle est un nombre exprimée en base 10 contenant un point décimal et éventuellement un exposant séparée du nombre par la lettre e ou E.

Exemples : 10. 5.4 3.5e-3.

Une constante caractère est assimilée à un entier codé sur un octet dont la valeur correspond au code ASCII du caractère. Elle est constituée d'un caractère entre apostrophes (quotes).

Exemples : 'z' 'A'

Une constante chaîne de caractères est une suite de caractères entre guillemets (double quote).

Exemple : "une chaîne". En mémoire cette suite de caractères se termine par le caractère NULL. Il s'agit en fait d'un tableau de caractères, notion que l'on abordera plus loin.

La directive #define n 100 permet de définir une constante s'appelant n et valant dans tout le programme 100.