[Math-L312] TP 5 : Typage en C

Adrien Semin
adrien.semin@math.u-psud.fr

1 Rappels sur les types du langage C

1.1 Les types de base

Les types de base correspondent au stockage de valeurs numériques. Dans le cas de valeurs non entières on parlera de nombres *flottants* car ceux-si sont représentés de façon interne en utilisant une écriture scientifique (mantisse et exposant) autorisant une position variable de la virgule : on l'appelle représentation à *virgule flottante*.

Types entiers		
Signés	Capacité ou taille	
Oui	Petite	`char`
	Moyenne	`short`
	Grande	`long`
Non	Petite	`unsigned char`
	Moyenne	`unsigned short`
	Grande	`unsigned long`

Types flottants	
Précision	
Simple	`float`
Grande	`double`
Encore plus grande	`long double`

Le type `int` usuel correspond au type entier signé natif le plus efficace pour les processeurs d'une machine donnée, qu'on désigne en général par le terme *mot*. Il correspond habituellement à `short`.

Les seules spécifications par rapport aux taille relatives des types d'une même famille entre eux, c'est que tous les nombres représentables par un type (entier ou flottant) doivent être représentables par le type de taille suivante (entier ou flottant). Par exemple, tout `short` peut être représenté par un `long`, mais en fonction de la machine où l'on travaille, ces 2 types peuvent être les mêmes.

1.2 Constantes littérales et tailles des types

Les constantes littérales numériques entières ou réelles suivent les conventions habituelles avec quelques particularités propres au langage C.

Les constantes entières

Elles sont sans signe : l'expression `-123` est comprise comme l'application de l'opérateur unaire `-` à la constante `123` ; mais puisque le calcul est fait pendant la compilation, cette subtilité n'a aucune conséquence pour le programmeur. Notez aussi qu'en C original, comme il n'existe pas d'opérateur `+` unaire, la notation `+123` est interdite.

Les constantes littérales entières peuvent aussi s'écrire en octal et en hexadécimal :

- une constante écrite en *octal* (base 8) commence par 0 (zéro) ;
- une constante écrite en *hexadécimal* (base 16) commence par `0x` ou `0X` (zéro, x).

Voici par exemple trois manières d'écrire le même nombre :

```
27      033      0x1B
```

Détail à retenir : on ne doit pas écrire de zéro non significatif à gauche d'un nombre : 0123 ne représente pas la même valeur que 123. Le type d'une constante entière est le plus petit type dans lequel sa valeur peut être représentée. Ou, plus exactement :

- si elle est décimale : si possible `int`, sinon `long`, sinon `unsigned long` ;

– si elle est octale ou hexadécimale : si possible `int`, sinon `unsigned int`, sinon `unsigned long`.

Certains suffixes permettent de changer cette classification :

– `U`, `u` : indique que la constante est d'un type `unsigned` ;
– `L`, `l` : indique que la constante est d'un type `long`.

Exemples : `1L` `0x7FFFU`
On peut combiner ces deux suffixes : `16UL`

Les constantes flottantes

Une constante littérale est l'expression d'un nombre flottant si elle présente, dans l'ordre :

– une suite de chiffres décimaux (la partie entière) ;
– un point, qui joue le rôle de virgule décimale ;
– une suite de chiffres décimaux (la partie fractionnaire) ;
– une des deux lettres `E` ou `e` ;
– éventuellement un signe `+` ou `-` ;
– une suite de chiffres décimaux.

Les trois derniers éléments forment l'exposant. Exemple : `123.456E-78`. On peut omettre :

– la partie entière ou la partie fractionnaire, mais pas les deux ;
– le point ou l'exposant, mais pas les deux.

Exemples : `.5e7` `5.e6` `5000000.` `5e6`

Une constante flottante est supposée de type `double`, à moins de comporter un suffixe explicite :

– les suffixes `F` ou `f` indiquent qu'elle est du type `float` ;
– les suffixes `L` ou `l` indiquent qu'elle est du type `long double`.

Exemples : `1.0L` `5.0e4f`

∗ ***Question orale*** ∗ A partir de là, **sans taper le code suivant**, prédire le résultat du bout de code suivant :

```
int a = 5;
float b = a / 2;
float c = a / 2.0;
```

Caractères et chaînes de caractères

Une constante de type caractère se note en écrivant le caractère entre apostrophes. Une constante de type chaîne de caractères se note en écrivant ses caractères entre guillemets.

Exemples, trois caractères : `'A'` `'2'` `'"'`

Quatre chaînes de caractères : `"A"` `"Bonjour à tous !"` `""` `"'"`

On peut faire figurer n'importe quel caractère, même non imprimable, dans une constante caractère ou chaîne de caractères en utilisant les combinaisons suivantes, appelées séquences d'échappement :

`\n`	Nouvelle ligne (LF)
`\t`	Tabulation (HT)
`\b`	Espace-arrière (BS)
`\r`	Retour-chariot (CR)
`\f`	Saut de page (FF)
`\a`	Signal sonore (BELL)
`\\`	\
`\'`	'
`\"`	"
$\backslash d_3d_2d_1$	Le caractère qui a pour code le nombre *octal* $d_3d_2d_1$; s'il commence par un ou deux zéros et si cela ne crée pas une ambiguïté, on peut aussi le noter $\backslash d_2d_1$ ou $\backslash d_1$.

Par exemple, la chaîne suivante définit la suite des 9 caractères[1] : A, *escape* (de code ASCII 27), B, ", C, saut de page, D, \ et E :

```
"A\033B\"C\fD\\E"
```

[1] En fait, le compilateur rajoute à la fin un caractère supplémentaire qui en marque la fin : le caractère \0.

Une constante de type caractère appartient au type char, c'est-à-dire entier représenté sur un octet. La valeur d'une constante caractère est le nombre qui représente le caractère de manière interne ; il s'agit presque toujours du code ASCII.

Une constante de type chaîne de caractères représente une suite finie de caractères, de longueur quelconque. Le codage interne d'une chaîne de caractères est le suivant :

- les caractères constituant la chaîne sont rangés en mémoire de manière contigüe, dans l'ordre où ils figurent dans la chaîne ;
- un caractère nul (`\0`) est ajouté après le dernier caractère pour indiquer la fin de la chaîne ;
- la constante chaîne représente alors, à l'endroit où elle est écrite, l'adresse de la cellule où est rangé le premier caractère.

Voici par exemple la représentation en mémoire (octet par octet) de la chaîne `"bonjour"` :

		b	o	n	j	o	u	r	\0		

Par conséquent, une constante chaîne de caractères a pour type celui d'un tableau de caractères (c'est à dire `char []`), et pour valeur l'adresse d'une cellule de la mémoire. Le caractère nul `\0` est celui dont le code ASCII est zéro[2]. On peut aussi le noter `0` en notation décimale.

∗ ***Question orale*** ∗ Pour la chaîne de caractères bonjour, on peut la stocker dans de la manière suivante :

```
char chaine[N]="Bonjour";
```

Quelle doit être la valeur minimale de N pour que le programme ne retourne par de `segmentation fault` ou de résultat incohérent à l'exécution du programme.

La taille des variables typées

Il ne faut pas oublier que d'une architecture machine à l'autre, la taille occupée par les différents types de base peut varier. La macro `sizeof` permet de connaître la taille mémoire occupée par un type ou une variable typée. On donne ici un programme en C qui permet d'afficher la taille occupée en octets par les types courants. Ce programme est téléchargeable à l'adresse `/home/doc/semin/TP/c/sizes.c`.

```
#include <stdio.h>

int main(){
  printf("sizeof(char) : %i\n",sizeof(char));
  printf("sizeof(short) : %i\n",sizeof(short));
  printf("sizeof(long) : %i\n",sizeof(long));
  printf("sizeof(unsigned char) : %i\n",sizeof(unsigned char));
  printf("sizeof(unsigned short) : %i\n",sizeof(unsigned short));
  printf("sizeof(unsigned long) : %i\n",sizeof(unsigned long));
  printf("sizeof(float) : %i\n",sizeof(float));
  printf("sizeof(double) : %i\n",sizeof(double));
  printf("sizeof(long double) : %i\n",sizeof(long double));
}
```

1.3 Les types dérivés

À partir des types de base précédents, on peut définir cinq nouveaux types qui en dérivent :

Les tableaux `[ ]`

Par exemple, on peut déclarer :

```
int T[255];
```

La variable `T` sera alors un tableau de `int` à 255 éléments. L'accès à ses éléments se fait avec des crochets (`[ ]`) : `T[0]` désigne le premier élément, `T[254]` le dernier. On peut aussi définir des types tableaux :

[2] Attention, il s'agit de `'\0'`, et surtout pas de `'0'` !

```
typedef float float_array[10000];
```

Cette ligne définit le type `float_array` : des tableaux de `float` à 10000 éléments. Il n'existe pas de tableaux multi-indicés en C, mais on peut toutefois faire des tableaux de tableaux : `int T[5][5]`.

Les déclarations de tableaux peuvent aussi être initialisées :

```
int T[5]={1,2,3,4,5};
```

Dans le cas de tableaux de caractères, on peut les initialiser avec une chaîne de caractères :

```
char S[10]="abcdefghi";
```

Le caractère `'\0'` sera ici inclus en bout de chaîne.

Les fonctions ( )

On peut définir des variables et des types de fonctions. Par exemple, dans ce programme téléchargeable à l'adresse `/home/doc/semin/TP/c/fun.c` :

```c
#include <stdio.h>

int f1(int x,int y){return x+y;} /* 2 "vraies" fonctions compatibles */
int f2(int x,int y){return x*y;}

int main(void){
  printf("Valeur: %i\n",f1(1,2));            /* Le résultat est 1+2=3 */
  return 0;
}
```

On déclare deux fonctions `f1` et `f2` qui prennent chacune deux arguments de type `int`, et qui retournent un argunement de type `int`.

On doit déclarer la fonction avant de l'utiliser. Pour améliorer la lisibilité du programme, on peut déclarer la fonction en disant quels types d'argument elle prend et quel type elle retourne, puis implémenter la fonction à la fin du programme, comme dans le code suivant, téléchargeable à l'adresse `/home/doc/semin/TP/c/fun2.c`

```c
#include <stdio.h>

int f1(int,int);

int main(void){
  printf("Valeur: %i\n",f1(1,2));            /* Le résultat est 1+2=3 */
  return 0;
}

int f1(int x,int y)
{return x+y;}
```

Programmation récursive

On parle de programmation récursive dans une fonction quand la fonction utilise elle-même, comma dans la code suivant, téléchargeable à l'adresse `/home/doc/semin/TP/c/funr.c`

```c
#include<stdio.h>

unsigned int add(unsigned int a,unsigned int b)
{
  if (b==0) return a;
  return add(a+1,b-1);
}

int main(void)
```

```
{
  printf("%d + %d = %d\n",3U,5U,add(3U,5U));
  return 0;
}
```

Le type `void`

Certaines fonctions peuvent très bien ne rien retourner. Dans ce cas-là, on les déclarera en void, comme dans le programme ci-dessous, téléchargeable à l'adresse `/home/doc/semin/TP/c/fun3.c`

```
#include<stdio.h>

void affichage(char[]);

int main(void)
{
  char chaine[200] = "Hello u-psud!\n";
  affichage(chaine);
  return 0;
}

void affichage(char t[])
{
  printf(t);
}
```

1.4 Principe du passage par valeur

Il est **très important** de bien comprendre que les variables passées comme valeur aux arguments aux fonctions ne sont pas modifiées. Un exemple :

```
#include <stdio.h>

int add(int x,int y){
  x=x+y;
  y=0;
  return x;
}

int main(){
  int x=3,y=2,z=0;
  z=add(x,y);
  printf("x=%d\n",x);
  printf("y=%d\n",y);
  printf("z=%d\n",z);
}
```

∗ ***Question orale*** ∗ Que va afficher ce programme si on l'exécute ?

2 Travail à rendre

1. `Approx.c` ∗ ***À Rendre!*** ∗ on demande de faire un programme qui initialise une variable de type `float` à `0.001`. Puis, à l'aide d'une boucle, on additionne successivement sa valeur dans une autre variable de type `float` initialisée à zéro, jusqu'à ce que la valeur de cette variable soit au moins égale à 1. Finallement, afficher le nombre d'itérations nécessaire.

   ∗ ***Question orale*** ∗ Quel phénomène explique le résultat obtenu ?

   ∗ ***Question orale*** ∗ Que se passe-t-il si l'on prend des variables de type `double` au lieu de `float` ?

2. `PowerDirect.c` ∗ ***À Rendre!*** ∗ Le programme suivant demandera un nombre réel x et un nombre entier p, et retourne le nombre réel x^p. On distinguera le cas $p > 0$ du cas $p < 0$ (pour le deuxième cas, on doit vérifier $x \neq 0$).

3. `PowerRecur.c` ∗ ***À Rendre!*** ∗ En remarquant le fait que $x^p = xx^{p-1}$ si p est positif, que $x^p = x^{p+1}/x$ si p est négatif, et avec la convention $x^0 = 1$, écrire le programme sous forme récursive.

4. `TableAscii.c` ∗ ***À Rendre!*** ∗ Ecrire un programme qui affiche tous les caractères dont le code ASCII est compris entre 32 et 127. Pour chaque caractère, on demandera un affichage du type

   ```
   Le caractère de code ASCII 32 est " "
   ```

   ∗ ***Question orale*** ∗ Que se passe-t-il si, au lieu de partir de 32, on part de 0 ?

5. `Alphabet.c` ∗ ***À Rendre!*** ∗ Le programme suivant demandera un caractère *lettre* et testera si la lettre est dans l'alphabet (on ne prendra en compte que les lettres non accentuées).

   ∗ ***Question orale*** ∗ Quel est le nombre maximum de tests à effectuer pour être sûr que la lettre est dans l'alphabet ?

   ∗ ***Question orale*** ∗ Pourquoi ne prenons nous pas en compte les lettres accentuées ?

6. `Uppercase.c` ∗ ***À Rendre!*** ∗ Ecrire une fonction qui prend en entrée une chaîne de caractères, et qui affiche la chaîne de caractères, dans laquelles les lettres minuscules non accentuées ont été remplacées par les lettres majuscules.