Séance 2 : Types et variables

Introduction

Tous les langages informatiques utilisent des variables. Une variable est un petit espace mémoire où l'on peut stocker une valeur qui pourra être modifiée au cours de l'exécution d'un programme. De plus le langage C est un langage dit « typé ». Ceci signifie qu'à chaque variable est associée un type qui va contraindre le type de variable qui pourra y être stockée.

Les types

Le langage C propose un certain nombre de types :

nom	*description*	*valeurs*	*espace mémoire*
`byte`	entier	-128...127	8 bits
`int`	entier	-32768...32767	16 bits
`long`	entier	-2147483648...2147483647	32 bits
`float`	flottant	$10^{-38}..10^{38}$, 6 chiffres de précision	32 bits
`double`	flottant	$10^{-38}..10^{38}$, 10 chiffres de précision	64 bits
`char`	caractère		8 bits

Les trois types les plus utilisés sont le type « `int` » pour les entiers, le type « `float` » pour les flottants et le type « `char` » pour les caractères.

Déclaration de variables

Une déclaration de variables se réalise comme suit :

```
type_de_la_variable  nom_de_la_variable;
```

Donc pour déclarer une variable de type « `int` » on pourra écrire :

```
int nombre;
```

La déclaration de variable se fait en début de bloc dans une fonction (et donc notamment dans la fonction « `main` ») et ceci avant tout appel de fonctions. Le nom d'une variable peut comporter tous les symboles alphanumériques non accentués et le caractère « _ », il doit de plus **commencer par une lettre**.
L'usage veut que les noms de variables soient explicites (comme pour celles que nous avons déjà définies) et écrit en minuscules. Si vous voulez que ce nom soit constitué de plusieurs mots vous pouvez les séparer par le caractère « _ » comme suit :

```
temperature_eau
```

Affectation d'une valeur à une variable

Pour affecter une valeur à une variable, on utilise la formulation suivante :

```
nom_de_la_variable  = expression;
```

Par exemple :

```
nombre    = 46;
somme     = 2.30;
```

Le signe « = » est ce que l'on appelle une opérateur. L'expression placée à gauche de cet opérateur est la variable sur laquelle on doit réaliser l'affectation. L'expression placée à droite est celle qui doit être affectée. Dans notre exemple, nous utilisons des expressions constantes pour l'affectation. Nous reviendrons sur cette notion d'opérateur un peu plus loin dans ce cours.

On peut de plus combiner ces expressions, ainsi la ligne :

```
a = b = c = 0;
```

affecte la valeur 0 aux trois variables a, b et c. Ceci n'est pas du au hasard. D'une part, l'expression « `a = 0` » a une valeur, celle de la valeur affectée à la variable. Et de plus ces expressions se regroupent de droite à gauche, la ligne équivaut donc à écrire :

```
a = ( b = ( c= 0));
```

On peut réaliser dans une même ligne de code la déclaration et l'affectation, c'est ce que nous allons voir dans le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
     int nombre     = 46;
     float somme    = 2.30f;
     char caractere = 'S';

     printf( "Variable nombre : %d\n", nombre);
     printf( "Variable somme : %f\n", somme);
     printf( "Variable caractere : %c\n", caractere);

     nombre = 2007;
     printf( "Variable nombre : %d\n", nombre);

     return(EXIT_SUCCESS);
}
```

Opérateurs arithmétiques

Nous venons de voir notre premier opérateur : l'opérateur « = » qui est un opérateur binaire. Les opérateurs binaires arithmétique sont `+`, `-`, `*`; `/` et `%` (reste de la division entière). Ces opérateurs sont utilisables indifféremment avec des types entiers ou flottants. L'opérateur `%` s'utilise uniquement avec des types entiers. Donc l'expression :

```
a * b
```

vaut le résultat de la multiplication de la variable a par la variable b.

Il existe deux opérateurs unaires : `+` et `-`. L'expression « `-a` » vaut l'opposé de la variable `a`.

Voici un exemple d'utilisation :

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
     double somme_en_francs   = 15.0;
     double somme_en_euros;

     somme_en_euros = somme_en_francs / 6.55957;

     printf( "Francs\t: %f\n", somme_en_francs);
     printf( "Euros\t: %f\n", somme_en_euros);

     return(EXIT_SUCCESS) ;
}
```

Les expressions

L'utilisation d'un opérateur constitue une expression, ainsi :

```
15 + 6
6.3 * conversion
-7
a = 12
```

sont des expressions. Les expressions ont une valeur, dans le cas des opérateurs arithmétiques cette valeur est bien sûr le résultat de l'opération désignée par l'opérateur. Dans le cas de l'opérateur d'affectation, le « = », on a vu que l'expression vaut la valeur affectée.

Autres opérateurs arithmétiques

Les opérateurs unaires ++ et -- permettent d'incrémenter ou de décrémenter une variable. L'expression :

```
a++;
```

ajoute 1 à la variable a.

Taper cet autre exemple de programme :

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
     int i;
     double somme_en_euros;
     double somme_en_francs;

     for( i=0; i<10; i++)
     {
          somme_en_euros = i;
          somme_en_francs     = somme_en_euros * 6.55957;
          printf( "%f euros valent %f francs\n",
               somme_en_euros, somme_en_francs);
     }
     return(EXIT_SUCCESS) ;
}
```

On peut aussi combiner les opérateurs arithmétiques et l'opérateur d'affectation. Ainsi toutes les expressions du type :

variable = variable opérateur expression;
exemple : `a = a * 4;`

équivaut à écrire :

variable opérateur = expression;
exemple : `a *= 4;`

mais une telle contraction d'écriture est déconseillée, car elle nuit à la lisibilité du programme

Priorité et associativité des opérateurs

Le tableau suivant classe les opérateurs par ordre décroissant de priorité.

opérateurs	*associativité*
`() [] -> .`	de gauche à droite
`! - ++ -- + - * & (type) sizeof`	de droite à gauche
`* / %`	de gauche à droite
`+ -`	de gauche à droite
`<< >>`	de gauche à droite
`< <= > >=`	de gauche à droite
`== !=`	de gauche à droite
`&`	de gauche à droite
`^`	de gauche à droite
`|`	de gauche à droite
`&&`	de gauche à droite
`||`	de gauche à droite
`?:`	de droite à gauche
`= += -= *= /= %= &= ^= |= <<= >>=`	de droite à gauche
`,`	de gauche à droite

Les opérateurs unaires +, − ont priorité sur leur forme binaire. De manière générale, il vaut mieux éviter de jouer sur les subtilités des priorités définies par le langage C, en utilisant autant que possible des parenthèses. Sinon, on risque de mauvaises surprises !

Exercice 1

Sachant qu'on a déclaré et intialisé des variables comme suit :

```
int a = b = 1;
int c = 32;
int d = 4;
```

Evaluer les expressions suivantes et la valeur des variables après l'exécution de ces expressions :

expression	*a*	*b*	*c*	*d*
`a = b++;`				
`b = c / 32 - b;`				
`a = (c-a)%d;`				
`b = a + (a = 3);`				

Exercice 2

Ecrire un programme qui donne le périmètre et l'aire d'un cercle pour deux rayons allant de 1 à 20.

Exercice 3

Faire un programme qui déclare deux entiers, leur affecte des valeurs et calcule puis affiche la division de l'un par l'autre, puis le reste de la division entière de l'un par l'autre.