Chapitre 1

LES GENERALITES DE LA PROGRAMMATION EN LANGAGE C

I. HISTORIQUE

Le C a été conçu en 1972 par Dennis Richie et Ken Thompson, chercheurs aux Bell Labs, afin de développer un système d'exploitation UNIX sur un DEC PDP-11. En 1978, Brian Kernighan et Dennis Richie publient la définition classique du C dans le livre « The C Programming language ». Le C devenant de plus en plus populaire dans les années 80, plusieurs groupes mirent sur le marché des compilateurs comportant des extensions particulières.
En 1983, l'ANSI (American National Standards Institute) décida de normaliser le langage ; ce travail s'acheva en 1989 par la définition de la norme ANSI C. Celle-ci fut reprise telle quelle par l'ISO (International Standards Organization) en 1990. C'est ce standard, ANSI C.

II. LA COMPILATION

Le C est un langage compilé (par opposition aux langages interprétés). Cela signifie qu'un programme C est décrit par un fichier texte, appelé fichier source. Ce fichier n'étant évidemment pas exécutable par le microprocesseur, il faut le traduire en langage machine. Cette opération est effectuée par un programme appelé compilateur. La compilation se décompose en fait en 4 phases successives :

II.1. *LE TRAITEMENT PAR LE PREPROCESSEUR*

Le fichier source est analysé par le préprocesseur qui effectue des transformations purement textuelles (remplacement de chaînes de caractères, inclusion d'autres fichiers source...).

II .2. *LA COMPILATION*

La compilation proprement dite traduit le fichier généré par le préprocesseur en assembleur, c'est-à-dire en une suite d'instructions du microprocesseur qui utilisent des mnémoniques rendant la lecture possible.

II.3. *L'ASSEMBLAGE*

Cette opération transforme le code assembleur en un fichier binaire, c'est-à-dire en instructions directement compréhensibles par le processeur. Généralement, la compilation et l'assemblage se font dans la foulée. Le fichier produit par l'assemblage est appelé fichier objet.

II.4. L'EDITION DE LIENS

Un programme est souvent séparé en plusieurs fichiers source, pour des raisons de clarté mais aussi parce qu'il fait généralement appel à des librairies de fonctions standard déjà écrites. Une fois chaque code source assemblé, il faut donc lier entre eux les différents fichiers objets. L'édition de liens produit alors un fichier dit exécutable.

Les différents types de fichiers utilisés lors de la compilation sont distingués par leur suffixe. Les fichiers source sont suffixés par .c, les fichiers prétraités par le préprocesseur par .i, les fichiers assembleur par .s, et les fichiers objet par .o. Les fichiers objets correspondant aux librairies précompilées ont pour suffixe .a.
Le compilateur C sous UNIX s'appelle **cc**. On utilisera de préférence le compilateur **gcc** du projet GNU. Ce compilateur est livré gratuitement avec sa documentation et ses sources. Par défaut, gcc active toutes les étapes de la compilation.
On le lance par la commande **gcc [options] fichier.c [-llibrairies]**

III. LES COMPOSANTS ELEMENTAIRES DU C

Un programme en langage C est constitué des six groupes de composants élémentaires suivants :

- **les identificateurs,**
- **les mots-clefs,**
- **les constantes,**
- **les chaînes de caractères,**
- **les opérateurs,**
- **les signes de ponctuation.**

On peut ajouter à ces six groupes les commentaires, qui sont enlevés par le préprocesseur lors de la compilation.

III.1. LES IDENTIFICATEURS

Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur
peut désigner :

- ✓ un nom de variable ou de fonction,
- ✓ un type défini par typedef, struct, union ou enum,
- ✓ une étiquette.

Un identificateur est une suite de caractères parmi :

- ✓ les lettres (minuscules ou majuscules, mais non accentuées),
- ✓ les chiffres,
- ✓ le « blanc souligné » **(_)**.

Le premier caractère d'un identificateur ne peut pas être un chiffre. Par exemple, **var1**, **tab_23** ou **_deb** sont des identificateurs valides ; par contre, 1i et i: j ne le sont pas. Il est cependant déconseillé d'utiliser **« _ »** comme premier caractère d'un identificateur car il est souvent employé pour définir les variables globales de l'environnement C.

ATTENTION : ***Les majuscules et minuscules sont différenciées.***

III.2. LES MOTS-CLEFS

Un certain nombre de mots, appelés mots-clefs, sont réservés pour le langage lui-même et ne peuvent pas être utilisés comme identificateurs. L'ANSI C compte 32 mots clefs.

auto	**const**	**double**	**float**	**int**	**short**	**struct**	**unsigned**
break	**continue**	**else**	**for**	**long**	**signed**	**switch**	**void**
case	**default**	**enum**	**goto**	**register**	**sizeof**	**typedef**	**volatile**
char	**do**	**extern**	**if**	**return**	**static**	**union**	**while**

III.3. LES COMMENTAIRES

Un commentaire débute par **/*** et se termine par ***/**.
Par exemple, //Seule la ligne est commentée
/* plusieurs lignes*/ Toutes les lignes comprises entre (/* et */) sont commentées
On ne peut pas imbriquer des commentaires. Quand on met en commentaire un morceau de programme, il faut donc veiller à ce que celui-ci ne contienne pas de commentaire.

IV. STRUCTURE D'UN PROGRAMME C

Un programme C se présente de la façon suivante :
[Directives au préprocesseur]
[Déclarations de variables externes]
[Fonctions secondaires]

```
main ()
 {
      Déclarations de variables internes ;
      Instructions ;
 }
```

La fonction principale **main** est la fonction principale et donc elle est obligatoire
Les fonctions secondaires peuvent être placées indifféremment avant ou après la fonction principale. Une fonction secondaire peut se décrire de la manière suivante :

```
type ma_fonction ( arguments )
 {
        déclarations de variables internes
        instructions
 }
```

V. LES TYPES PREDEFINIS

Le C est un langage *typé*. Cela signifie en particulier que toute variable, constante ou fonction est d'un type précis. Le type d'un objet définit la façon dont il est représenté en mémoire. La mémoire de l'ordinateur se décompose en une suite continue d'octets. Chaque octet de la mémoire est caractérisé par son adresse, qui est un entier. Deux octets contigus en mémoire ont des adresses qui diffèrent d'une unité. Quand une variable est définie, il lui est attribué une adresse. Cette variable correspondra à une zone mémoire dont la longueur (le nombre d'octets) est fixée par le type.
Les types de base en C concernent les caractères, les entiers et les flottants (nombres réels). Ils sont désignés par les mots-clefs suivants :
char int float double short long unsigned

V.1. LE TYPE CARACTERE

Le mot-clef ***char*** désigne un objet de type caractère. Un char peut contenir n'importe quel élément du jeu de caractères de la machine utilisée. La plupart du temps, un objet de type char est codé sur un octet ; c'est l'objet le plus élémentaire en C.
Une des particularités du type char en C est qu'il peut être assimilé à un entier : tout objet de type char peut être utilisé dans une expression qui utilise des objets de type entier. Par exemple, si c est de type char, l'expression c + 1 est valide. Elle désigne le caractère suivant dans le code ASCII. Ainsi, le programme suivant imprime le caractère 'B'.

```
main()
    {
      char c = 'A';
      printf("%c", c + 1);
    }
```

V.2. LES TYPES ENTIERS

Le mot-clef désignant le type entier est ***int***. Le type int peut être précédé d'un attribut de précision (*short* ou *long*) et/ou d'un attribut de représentation (*unsigned*).Un objet de type short int a au moins la taille d'un char et au plus la taille d'un int. En général, un short int est codé sur 16 bits. Un objet de type long int a au moins la taille d'un int (64 bits sur un DEC alpha, 32 bits sur un PC Intel).

V.3. LES TYPES FLOTTANTS

Les types ***float***, double et long double servent à représenter des nombres en virgule flottante (décimaux). Ils correspondent aux différentes précisions possibles.

VI. LES OPERATEURS

VI.1 L'AFFECTATION

En C, l'affectation est un opérateur à part entière. Elle est symbolisée par le signe égal **(=)**. Sa syntaxe est la suivante :

variable = expression ;

Le terme de gauche de l'affectation peut être une variable simple, un élément de tableau mais pas une constante.
Cette expression a pour effet d'évaluer expression et d'affecter la valeur obtenue à variable. De plus, cette expression possède une valeur, qui est celle expression. Ainsi, l'expression i = 5 vaut 5.
L'affectation effectue une conversion de type implicite : la valeur de l'expression (terme de droite) est convertie dans le type du terme de gauche. Par exemple, le programme suivant

```
main()
{
int i, j = 2;
float x = 2.5;
i = j + x;
x = x + i;
printf("\n %f \n",x);
}
```

imprime pour x la valeur 6.5 (et non 7), car dans l'instruction i = j + x;, l'expression j + x a été convertie en entier.

VI.2. LES OPERATEURS ARITHMETIQUES

Les opérateurs arithmétiques classiques sont l'opérateur unaire - (changement de signe) ainsi que les opérateurs binaires

+ addition
- soustraction
*** multiplication**
/ division
% reste de la division (modulo)

Ces opérateurs agissent de la façon attendue sur les entiers comme sur les flottants. Leurs seules spécificités sont les suivantes :
Contrairement à d'autres langages, le C ne dispose que de la notation / pour désigner à la fois la division entière et la division entre flottants. Si les deux opérandes sont de type entier, l'opérateur / produira une division entière (quotient de la division). Par contre, il délivrera une valeur flottante dès que l'un des opérandes est un flottant.
Par exemple,

```
float x;
x = 3 / 2;
```

affecte à x la valeur 1. Par contre

```
x = 3 / 2.;
```

affecte à x la valeur 1.5.
L'opérateur % ne s'applique qu'à des opérandes de type entier. Si l'un des deux opérandes est négatif, le signe du reste dépend de l'implémentation, mais il est en général le même que celui du dividende.
Notons enfin qu'il n'y a pas en C d'opérateur effectuant l'élévation à la puissance. De façon générale, il faut utiliser la fonction **pow(x,y)** de la librairie **math.h**
pour calculer x^y.

VI.3. Les opérateurs relationnels

> supérieur à
>= supérieur ou égal à
< strictement inférieur à
<= inférieur ou égal à
= = égal à
!= différent de

Leur syntaxe est **: expression1 op expression2**
Les deux expressions sont évaluées puis comparées. La valeur rendue est de type int (il n'y a pas de type booléen en C); elle vaut 1 si la condition est vraie, et 0 sinon.
NB : ***à ne pas confondre l'opérateur de test d'égalité = = avec l'opérateur d'affection =.***

```
main()
 {
   int a = 0;
   int b = 1;
   if (a = b)
   printf("\n a et b sont égaux \n");
   else
   printf("\n a et b sont différents \n");
}
```

Imprime à l'écran a et b sont égaux !

VI.4. LES OPERATEURS LOGIQUES BOOLEENS

&& et logique

|| ou logique

! négation logique

Comme pour les opérateurs de comparaison, la valeur retournée par ces opérateurs est un int qui vaut 1 si la condition est vraie et 0 sinon.