Chapitre 1 : Initiation au langage C : Notions de base

Objectifs

- *Introduction au langage C*
- *Connaître la structure d'un programme C*
- *Connaître les composantes élémentaires du C*
- *Connaître les opérateurs du langage C*
- *Connaître les principaux types de donnés du langage C*

1. Introduction au langage C

1.1. 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 l983, 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 qu'elle par l'ISO (International Standards Organization).

1.2. Présentation générale du langage C

Le langage **C** est un langage évolué et structuré, assez proche du langage machine destiné à des applications de contrôle de processus (gestion d'entrées/sorties, applications temps réel ...). Les compilateurs C possèdent les taux d'expansion les plus faibles de tous les langages évolués (rapport entre la quantité de codes machine générée par le compilateur et la quantité de codes machine générée par l'assembleur et ce pour une même application);

Le langage C possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur.

Exemples: math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

On ne saurait développer un programme en C sans se munir de la documentation concernant ces bibliothèques.

2. Structure d'un programme C

Un programme en C comporte :

- ***Un entête*** (header) constitué de méta instructions ou **directives** destinées au pré processeur.

- ***Un bloc principal*** appelé **main ().** Il contient des instructions élémentaires ainsi que les appels aux modules (fonctions) considérés par le compilateur comme des entités autonomes. Chaque appel à une fonction peut être considéré comme une instruction.
- ***Le corps*** des fonctions placées avant ou après le main () dans un ordre quelconque, les une après les autres. Partout, les variables et les fonctions font l'objet d'une déclaration précisant leur type. Le schéma général est donc :

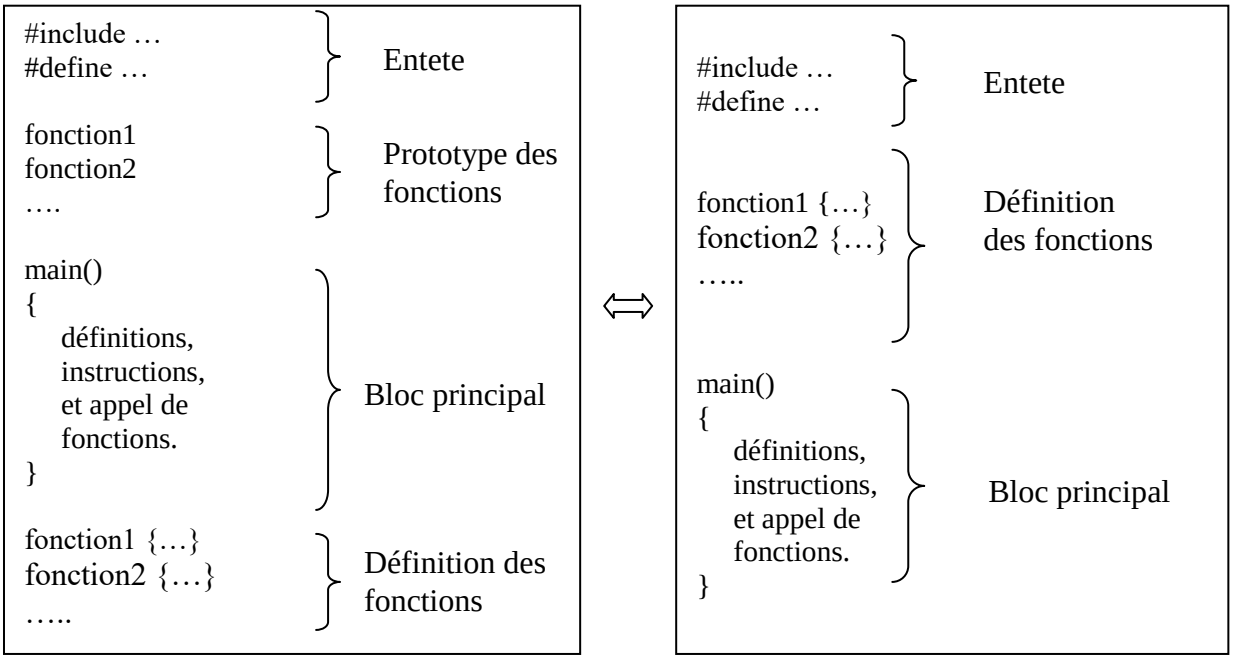

Figure 1.1 : structure d'un programme C

Exemple : calcul de la surface d'un cercle.

```
#include <stdio.h>
#define PI 3.14
float f_surface(float rayon)
{float s;
 s=rayon*rayon*PI;
 return(s);
}
void main()
{  float surface;
surface=f_surface(2.0);
printf("%f\n",surface);
}
```

```
#include <stdio.h>
#define PI 3.14
float f_surface(float);
void main()
{
float surface;
surface=f_surface(2.0);
printf("%f\n",surface);
}
float f_surface(float rayon)
{float s;
 s=rayon*rayon*PI;
 return(s);
}
```

3. Les composants élémentaires 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 ne sont pas considérés par le pré processeur.

3.1. Les identificateurs

- Le rôle d'un identificateur est de donner un nom à une entité du programme. Plus précisément, un identificateur peut designer :
  - 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 tirait bas.
- Le premier caractère d'un identificateur ne peut pas être un chiffre.
- Les majuscules et minuscules sont différenciées.
- Le compilateur peut tronquer les identificateurs au-delà d'une certaine longueur. Cette limite dépend des implémentations, mais elle est toujours supérieure à 31 caractères. (Le standard dit que les identificateurs externes, c'est-à-dire ceux qui sont exportés à l'édition de lien, peuvent être tronqués à 6 caractères, mais tous les compilateurs modernes distinguent au moins 31 caractères).

3.2. 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

3.3. Les commentaires

Un commentaire débute par /* et se termine par */. Par exemple : /* Ceci est un commentaire */. On ne peut pas imbriquer des commentaires.

3.4. Les constantes

Une constante est une valeur qui apparaît littéralement dans le code source d'un programme, le type de la constante étant déterminé par la façon dont la constante est écrite. Les constantes vont être utilisées, par exemple, pour l'initialisation de variables.

Exemples :

```
int NB_ligne ;          /* déclaration d'une variable de type entière */
NB_ligne = 15 ;         /*  initialisation de la variable à 15 */
const annee = 2005  /* affecte une valeur à une variable, cette valeur ne pourra plus être changé */
```

4. Les types de données

En C, il n'existe que quelques types fondamentaux de données :

- Le type **char :** un seul byte représentant un caractère.
- Le type **int :** un nombre entier dont la taille corresponds à celles des entiers du SE.
- Le type **float :** un nombre en virgule flottante en simple précision.
- Le type **double :** un nombre en virgule flottante en double précision.

Des qualificatifs peuvent préciser le type int : **short, long, unsigned, signed**. Les qualificatifs **signed** et **unsigned** peuvent préciser le type char. Le qualificatif **long** peut préciser le type double. Quand une déclaration ne précise pas le type de base, **int** est supposé.

5. Déclaration de variables

Les variables doivent toutes être déclarées avant d'être utilisées bien que certaines soient faîtes implicitement par le contexte. A une variable correspond un espace en mémoire et un mécanisme permettant d'adresser cet emplacement. En C une variable est caractérisé par :

- Son nom (un identificateur)
- Son type (type de base ou type définie par l'utilisateur)
- Sa classe d'allocation (extern, static, auto, register)

Une déclaration de variables a la forme :

type liste d'identificateurs

Exemples :

```
short int x ;
unsigned long y ;  /* int implicite */
long double z ;
```

Une déclaration indique le nom, le type (parfois la valeur) et regroupe derrière une ou plusieurs variables de même type.

On peut initialiser les variables lors de la déclaration :

Exemples :

```
int i = 0 ;
```

6. Les opérateurs

6.1. L'affectation

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

variable = expression ;

Le terme de gauche de l'affectation peut être une variable simple, un élément de tableau, …etc. 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 de « 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("%f",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.

6.2. Les opérateurs arithmétiques

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) (pour les entiers)

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 :
- 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

6.3. Les opérateurs relationnels

> Strictement supérieur ; >= supérieur ou égal ; < Strictement inférieur ;
<= inférieur ou égal ; == égal ; != différent

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 0 si la condition est faux, et une valeur différente de 0 sinon. Attention à ne pas confondre l'opérateur de test d'égalité == avec l'opérateur d'affection =. Ainsi, le programme suivant imprime à l'écran : a et b sont égaux .:

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

6.4. Les opérateurs logiques (booléens)

&& : et logique || : ou logique ! : négation logique

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

Dans une expression de type : expression1 op1 expression2 op2 ...expressionN , l'évaluation se fait de gauche à droite et s'arrête dès que le résultat final est déterminé.

6.5. Les opérateurs d'affectation composée

Les opérateurs d'affectation composée sont : += -= *= /= %= &= ^= |= <<= >>=

Pour tout opérateur **op**, l'expression :

expression1 **op=** expression2 ⇔ expression1 = expression1 **op** expression2

Toutefois, avec l'affection composée, expression1 n'est évaluée qu'une seule fois.

6.6. Les opérateurs d'incrémentation et de décrémentation

Les opérateurs d'incrémentation ++ et de décrémentation -- s'utilisent aussi bien en suffixe (i++) qu'en préfixe (++i). Dans les deux cas la variable i sera incrémentée, toutefois dans la notation suffixe la valeur retournée sera l'ancienne valeur de i alors que dans la notation préfixe se sera la nouvelle. Par exemple :

```
int a =3, b, c ;
b = ++a ;          /* a et b valent 4 */
c = b++ ;          /* c vaut 4 et b vaut 5 */
```

6.7. L'opérateur conditionnel ternaire

L'opérateur conditionnel **?** est un opérateur ternaire. Sa syntaxe est la suivante :

```
Condition ? expression1 : expression2
```

Cette expression est égale à expression1 si condition est satisfaite, et à expression2 sinon. Par exemple, l'expression x >= 0 ? x : -x correspond à la valeur absolue d'un nombre. De même l'instruction m = ((a > b) ? a : b); affecte à m le maximum de a et de b.

6.8. L'opérateur de conversion de type

L'opérateur de conversion de type, appelé **cast**, permet de modifier explicitement le type d'un objet. On écrit **(type) objet** Par exemple : retourne la valeur 1.5.

```
int i = 3, j = 2 ;
printf("%f \n", (float)i/j) ;
```

6.9. L'opérateur adresse

L'opérateur d'adresse **&** appliqué à une variable retourne l'adresse mémoire de cette variable. La syntaxe est **&objet .**

7. Les types prédéfinis

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

7.1. Les types caractères

Le mot-clef char désigne un objet de type caractère. Il est codé sur un octet et 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.

Exemple : le programme suivant affiche la lettre B.

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

7.2. Les types entiers

Un objet de type entier : int est représenté par un mot de 32 bits pour un PC Intel.

Le type int peut être précédé d'un attribut de précision (short ou long) et/ou d'un attribut de représentation (unsigned). Les valeurs maximales et minimales des différents types entiers sont définies dans la librairie standard limits.h. Le mot-clef sizeof a pour syntaxe sizeof(expression) où expression est un type ou un objet. Le résultat est un entier égal au nombre d'octets nécessaires pour stocker le type ou l'objet.

7.3. Les types flottants

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