Programmation en c

I. Introduction :

I .1- Structure d'un programme C

Un programme C est un texte écrit dans une norme stricte qui respecte ce que l'on appelle une

syntaxe. La structure d'un programme C est la suivante :

<instructions d'entête du programme>

<fonction 1>

<fonction2>

. ..

...

<fonction n>

Une fonction est un bloc indépendant qui porte un nom appelé nom de la fonction et qui est décrit relativement à une syntaxe précise. Ce bloc correspond à un traitement aboutissant à une tâche définie.

Les fonctions que l'on décrit dans un programme C sont des fonctions utilisateurs. Elles sont spécifiques au programme. Toutefois, il existe des fonctions cataloguées qui appartiennent à la bibliothèque du compilateur C ; ces fonctions existent déjà, le programme peut les appeler.

Une fonction peut appeler une autre fonction qu'elle soit utilisateur ou appartenant à la bibliothèque du C.

Pour appeler une fonction dans un programme C, il suffit de préciser son nom dans une instruction et de lui communiquer les paramètres utiles. Nous verrons plus tard la façon avec laquelle on développe des fonctions en C.

Parmi les fonctions d'un programme C, il doit exister une qui porte le nom **main**.

La fonction main est la première fonction exécutée au lancement d'un programme C.

Un programme C peut se limiter à une seule fonction, ça serait dans ce cas un texte contenant la seule fonction main.

Les instructions d'entête d'un programme C correspondent à des définitions réclamées par les fonctions de la bibliothèque standard du C ou des définitions qui correspondent à des fonctions utilisateurs. On se limitera pour l'instant à introduire des définitions correspondantes à des inclusions de fichiers d'entête du compilateur C.

Un fichier d'entête est un fichier avec une extension .h et qui est inclus dans un programme pour les définitions utiles aux fonctions évoquées dans un programme.

L'exemple suivant illustre le texte d'un programme C qui a comme tâche d'afficher sur l'écran le message : C'est mon premier test de programmation C

```
#include <stdio.h>
main()
{
printf("C'est mon premier test de programmation C");
}
```

Remarques:

1°) Ce programme est donc un texte qui contient 5 lignes.

2°) Les lignes

```
main()
{
printf("C'est mon premier test de programmation C");
}
```

constituent le texte de la fonction main.

3°) Le caractère { représente une marque de début de bloc et le caractère } représente une marque de fin de bloc .

4°) printf est une fonction de la bibliothèque standard du C qui permet d'écrire un message formaté.

5°) La ligne

#include <stdio.h>

permet d'inclure un fichier de définitions qui s'appelle stdio.h (standard input output). ce ci est nécessaire à chaque fois que l'on utilise dans un programme un appel à une fonction de lecture ou d'écriture (et c'est le cas dans ce programme puisque on fait appel à la fonction printf) .

6°) La ligne d'entête commence par le caractère #, c'est pour dire que c'est une instruction du

pré-compilateur C. En effet les lignes qui commencent par # seront convertis par le compilateur en fonction de leur nature, le #include permet d'informer le pré-compilateur C d'inclure le fichier stdio.h dans le programme.

I.2 - Structure de la fonction main

En C, la fonction main, représentant le programme principal, est composé de deux parties :

- une partie déclarative : réservée à la déclaration des variables,

- une partie traitement : contenant les instructions qui manipulent les objets déclarés.

```
main()
{
<partie déclarative>
<partie traitement>
}
```

II. Notion de variables et de constantes

II .1 Les variables

➔Un programme a besoin de gérer de l'espace mémoire pour appliquer un traitement.

➔A chaque fois que l'on souhaite utiliser une zone mémoire, il y a lieu de définir dans le programme C une zone que l'on appelle variable.

➔Une variable mémoire correspond donc à une zone allouée par le programme et dans laquelle il peut placer une valeur.

➔La valeur correspond à une expression ou à une constante qui dépend du type de la variable.

➔Une variable est définie par son nom et par son type.

II .2 Les constantes

Une constante correspond à une valeur d'un type donné. Contrairement à la variable qui désigne le contenu d'une zone mémoire qui peut changer durant l'exécution d'un programme, une constante correspond à une valeur fixe.

Les constantes différent du type des valeurs qu'elles désignent.

II.3 Les types de Constantes

Les constantes en C se distinguent par rapport au type de données.

Pour chaque type, le langage C a prévu des constantes qui devront être utilisés par le programme en concordance avec les variables impliquées par les affectations.

• Constantes de type int :

Ces constantes correspondent à des valeurs numériques positives ou négatives.

Exemple de constantes de type int :

0

1200

432

-12

Remarque :

Comme nous l'avons déjà précisé, la déclaration d'une variable mémoire permet d'allouer un espace mémoire d'une taille donnée. Une constante int doit être donc en concordance avec la taille de l'espace mémoire relativement au type **int**.

Si la taille d'une variable mémoire de type int est de 2 octets (16 bits), et sachant que le premier bit de cet espace est réservé pour le signe, il ne reste que 15 bits pour la représentation de la valeur. La valeur maximale que l'on peut donc placer dans un tel espace est celle d'une valeur en binaire formée par 15 bits tous égaux à 1 et qui correspond à la valeur : 32767.

La constante 45000 par exemple est donc non valide.

Il en est de même pour la constante –70000.

Ces deux constantes ne peuvent pas être représentées sur 2 octets et seront rejetées par le compilateur C.

• Constantes de type long

Ces constantes se réfèrent au type long dont la taille en nombre d'octets qu'il prévoit pour la zone mémoire est le double que celle du type int.

Ces constantes sont comme celles évoquées pour le type int mais qui se termine avec la lettre L.

Exemple :

0L

70L

70000L

-145000L

Pour une taille de 4 octet associée au type long, la valeur maximale que l'on peut placer dans une variable de type long est donc celle qui correspond à une valeur en binaire formée par 31 bits tous égaux à 1 (le premier bit correspond au bit de signe).

Ainsi, la constante 70000L est donc valide puisque on peut la représenter sur 4 octets.

• Constante de type char

Ces constantes correspondent aux caractères alphanumériques et aux caractères spéciaux.

Ces caractères sont précédés et suivis par le symbole ‘.

Les constantes de type char sont donc les caractères :

• Alphabétiques minuscules : ‘a’ ‘b’ ‘c’ ..... ‘z’

• Alphabétiques majuscules : ‘A’ ‘B’ ‘C’ ..... ‘Z’

• Les chiffres : ‘ ‘ ‘0’ ‘1’ ‘2’ .... ‘9’

• les caractères spéciaux : tels que‘.’ ‘,’ ‘ ;’ ‘#’ ,’ !’, ‘ ?’ ‘+’ ‘-‘ ....

Pour chaque constante de type char correspond une valeur numérique qui représente le code du caractère. Ce code découle du système de représentation des caractères et qui est généralement le code ASCII qui est une norme standard de la représentation des caractères.

Pour chaque caractère correspond donc un code.

Exemple :

En ASCII

Le code de ‘A’ est 65

Le code de ‘B’ est 66

Le code de ‘a’ est 97

Le code de ‘b’ est 98

Le code de ‘0’ est 48

Le code de ‘1’ est 49

Le code de ‘ ‘ est 32

La taille pour une variable de type char est généralement de 1 octet. Cet espace suffit pour représenter le code d’une constante de type caractère dont le nombre est fixe et ne dépassant pas 255 (pour les caractères usuels).

• **constante de type float**

Ces constantes sont prévues pour les réels.

Le langage C gère les réels en représentation flottante (d’où le terme float).

Le point est utilisé pour séparer la partie entière de la partie décimale.

Exemple de constantes de type float :

12.5

1.543000008

0.

-1.

1 E-5 (pour designer la valeur (0.00001)

La taille d'un flottant est en général de 4 octets. Il est à noter qu'il n'y a pas une taille fixe en nombre de bits pour la composante entière et pour la composante décimale, la virgule étant flottante. Toutefois pour gérer des valeurs d'une grande précision il convient d'utiliser le type double qui permet une représentation en double précision.

• Les Constantes de type double

La représentation des constantes de type double est équivalente à celle des constantes float mais elle peut se référer à des valeurs supérieures à celles que l'on peut représenter pour le type float, étant donné que la taille en nombre d'octets est le double du float.

II.4 Instruction de déclaration

II.4. 1 Déclaration simple

➔Pour déclarer une variable on fait appel à une instruction de déclaration.

➔La syntaxe d'une instruction de déclaration est la suivante :

<type> <nom de la variable> ;

les types standards du langage C sont :

int : pour les entiers

long : pour les entiers longs

char : pour les caractères

float : pour les réels

double: pour les réels en double précision.

Un nom d'une variable est formé par un mot qui commence par une lettre alphabétique ou le caractère _ , suivi par des lettres ou des chiffres ou le caractère _.

Exemple :

int premier ;

Cette instruction permet de réserver au programme une zone appelée premier et qui est de type int.

II.4.2 Déclaration multiple

Syntaxe :

<type> <liste de variables> ;

La liste des variables correspond à des noms de variables séparées par une virgule.

Exemple :

int premier, dernier, total, i1, i2 ;

• Taille de l'espace mémoire réservé :

L'instruction de déclaration permet de réserver un emplacement mémoire d'une taille en

nombre d'octets fixe. La taille de la zone allouée dépend du type de la variable et aussi du compilateur C que l'on utilise.

Pour le développement des exemples , on retient une configuration où la correspondance type et taille est la suivante :

int : 2 Octets

long : 4 Octets

char : 1 Octet

float : 4 Octets

double : 4 octets

III TRAITEMENT SEQUENTIEL

III.1 - Introduction

Pour pouvoir résoudre un problème, nous devons décrire la solution à travers des actions ordonnées manipulant des objets. En fait, ces actions sont représentées par des instructions donnant l'ordre à un processeur pour effectuer des tâches bien déterminées allant des tâches les plus simples vers les plus complexes. Nous allons présenter progressivement dans la suite de ce cours ces instructions.

III.2 Instruction d'affectation

Une instruction d'affectation permet de placer dans une zone mémoire une valeur.

La zone mémoire désignée par cette opération correspond à une variable mémoire

préalablement déclarée.

La valeur placée dans la zone mémoire est la valeur de l'évaluation d'une expression.

La syntaxe d'une instruction d'affectation est la suivante :

<variable> = <expression> ;

l'opérateur = est appelé opérateur d'affectation.

Exemple :

int i ;

float total, initial ;

```
char c;
I =2;
c='M';
initial = 3400. ;
total = (initial –100.) * i;
```

III.3 Les instructions d'entrée/sortie

• Fonction de lecture d'un caractère

Pour accepter un caractère du clavier, le langage C dispose d'une fonction qui permet de récupérer le code du caractère sur lequel l'utilisateur a tapé. Cette fonction est getchar()

Exemple :

```
char c ;
c=getchar() ;
```

La fonction getchar() génère une interruption du programme jusqu'à ce que l'utilisateur tape sur une touche du clavier, elle renvoie dans ce cas le code ASCII du caractère correspondant.

Remarque :

L'utilisation de la fonction getchar() (ainsi que toutes les fonctions de lecture et d'écriture) nécessite l'inclusion du fichier stdio.h dans l'entête du programme.

Voici un exemple d'un programme C qui accepte dans une zone mémoire un caractère du clavier :

```
#include <stdio.h>
main()
{
char c;
c=getchar();
}
```

• Fonction d'écriture d'un caractère

La fonction qui permet d'afficher un caractère sur l'écran est la fonction putchar().

Cette fonction admet comme paramètre le code du caractère que l'on doit afficher et qui est fournit à travers une variable de type char ou d'une constante de type char.

Le programme suivant permet de lire à partir du clavier un caractère et de l'afficher sur l'écran.

```
#include <stdio.h>
main()
{
char c;
c=getchar();
putchar(c);
}
```

• Fonction de Lecture formatée

Introduction :

La fonction getchar() que l'on vient de présenter permet de lire du clavier un caractère. C'est une fonction spécifique pour le type char. Si le programme a besoin de lire une donnée d'un autre type il devra faire appel à une fonction qui se charge de récupérer une suite de caractères et de les convertir au type voulu.

La fonction scanf() permet de lire des valeurs d'un type donné.

Syntaxe :

La syntaxe d'utilisation de la fonction scanf() est la suivante :

scanf (< format > , < adresse 1> [<adresse 2> .....]) ;

< format > : est une chaîne de caractères incluant des spécifications de conversions.

Une spécification de conversion est introduite par l'opérateur %.

On distingue :

• %c pour lire un caractère

• %d pour lire un entier

• %ld pour lire un long

• %f pour lire un float

• %lf pour lire un double

< adresse i > : est l'adresse d'un argument (variable)

Exemple

soient les déclarations suivantes:

int i ;

char a ;

float r ;

l'appel suivant:

scanf("%d",&i);

permet de lire une valeur de type entier et de la placer dans l'adresse mémoire associée à la variable i.

L'opérateur & évoque l'adresse d'une variable.

&i est donc l'adresse de la variable i.

Si à l'exécution l'utilisateur donne 430, alors la fonction scanf() va donc lire les caractères 4, puis 3, puis 0, et de les convertir en une valeur 430 et de placer cette valeur dans l'adresse de la variable i. l'appel :

scanf("%f ",&r) ;

permet de lire un réel et de le placer dans l'adresse de la variable r.

l'appel :

scanf(" %d %c %f " , &i , &a , &r) ;

permet de lire un entier, un caractère et un réel.

Supposons qu'à l'exécution du programme, l'utilisateur introduit la chaine de caractères suivante :

1250 w 12.53

les affectations seront alors :

• la valeur 1250 pour la variable i,

• le caractère 'w' pour la variable a,

• et la valeur 12.53 pour la variable r.

Remarque :

Une spécification de conversion peut fixer le nombre de caractères à prendre en considération

dans la lecture et la conversion.

Exemple :

int i , j , k ;

scanf(" %3d %2d %4d " , &i , &j , &k);

Supposons qu'à l'exécution du programme l'utilisateur introduit la chaine de caractères suivante :

123706535

L'exécution de la fonction scanf() va générer l'affectation de :

• la valeur 123 pour la variable i,

• la valeur 70 pour la variable j,

• et la valeur 6535 pour la variable k.

• Fonction d'écriture formatée

Introduction :

La fonction d'écriture putchar() permet d'écrire un seul caractère.

Pour afficher sur l'écran une valeur d'un autre type on a besoin d'une fonction qui prend en considération le type de la valeur et la transforme en chaîne de caractères avant de l'afficher sur l'écran. C'est la fonction printf() qui permet d'effectuer ce travail.

Syntaxe :

La syntaxe de la fonction printf() est la suivante :

printf(< format > , < argument 1> [< argument 2> .....])

< format > : est une chaîne de caractères incluant des spécifications de conversions. Une spécification de conversion est introduite par l'opérateur %. Elle sert pour convertir des valeurs d'un type donné en suite de caractères.

< argument i > : est une variable

Exemple :

soient les déclarations suivantes:

```
int i ;
float r ;
i = 25;
r = 12.704
printf ( " Le résultat : %2d %7.3f " , i , r ) ;
```

L 'exécution de la fonction printf() génère l'affichage (sur écran) de la chaîne de caractères suivante :

Le résultat :25 12.704

Le format %7.3f permet de spécifier que le réel est à convertir sous une chaîne de longueur totale égale à 7 dont 3 caractères pour la partie décimale.