Travaux Pratiques N°1
Introduction à la programmation C

Objectifs :

- Comprendre la structure d'un programme C
- Evoquer les principales fonctions des menus déroulants (File, edit, run, compile…)
- Etudier les étapes de compilation, exécution et test
- Ecrire et exécuter un programme qui permet d'afficher une chaine de caractère sur l'écran
- Détecter et corriger les erreurs syntaxiques d'un programme

I. Introduction

Le langage C est un langage évolué et structuré, assez proche du langage machine destiné des applications de contrôle de processus (gestion d'entrées/sorties, application temps réel…).
Le langage C possède un peu d'instructions. Il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombres avec le compilateur.
Exemple : math.h : bibliothèque de fonctions mathématiques
stdio.h : bibliothèque d'entrées/sorties standard

II. Squelette d'un programme C

On peut définir le squelette d'un programme C de la façon suivante :
Déclaration des bibliothèques
Déclaration des constantes
main ( ){
Déclaration des variables
Corps du programme
}

Remarques :

Il est préférable de commenter les instructions difficiles. Ceci permet d'expliquer la signification de ces dernières au lecteur même s'il ne connaît pas le langage de programmation. Les commentaires en C peuvent être placés à n'importe quel endroit du programme. Ils commencent par /* et se terminent par */ sur une même ligne.
Le langage C distingue les minuscules, des majuscules. Les mots réservés du langage C doivent être écrits en minuscules.
Il est préférablede prendre l'habitude d'encadrer le corps du programme de la façon suivante :

```
#include<stdio.h> /*déclaration de la bibliothèque d'E/S standard */
#include<conio.h> /* déclaration de la bibliothèque qui  contient clrscr () et getch ()*/

void main ( ){
   /*déclaration des variables*/
clrscr ( ) ; /*facultatif, permet d'effacer l'écran à chaque exécution*/

   /*corps du programme*/
getch  ( ) ; /*facultatif, permet de voir ce qui est produit à l'écran en*/
            /*attendant l'appui d'une touche*/
}
```

III. Identificateurs et mots-clés

Un identificateur est le nom donné à une variable, une constante, une fonction, etc. Ce nom est composé de lettres non accentuées, de chiffres et du caractère dans un ordre quelconque sauf pour le premier caractère qui ne peut pas être un chiffre. Certains mots ne peuvent pas être utilisés comme identificateurs car ils sont réservés, ce sont des mots-clés du langage. Ceux-ci ne vous diront rien pour le moment, tout ce qu'il faut savoir c'est que vous ne devez pas utiliser ces mots pour nommer vos variables et fonctions : **auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while**.

IV. Variables

Une variable est un espace mémoire que le programme réserve lors de son exécution au moment de sa déclaration. C'est une donnée dont la valeur est modifiable.

V. Déclaration de variable

Lorsqu'on programme, il faut toujours donner le type des variables avant de les utiliser. C'est ce qu'on appelle la *déclaration*.

Syntaxes :

```
nom_typenom_variable;
nom_type nom_variable_1, nom_variable_2, ..., nom_variable_N ;
```

Exemples :

```
int A ; /*déclaration d'une variable de type entier*/
float X, Y ; /*déclaration de deux variables de type réel*/
```

Affectation de variable

Le symbole <- que nous utilisions en algorithmique correspond au signe = du langage C.

Syntaxe :

```
nom_variable = valeur;
```

Exemple :

```
int x, y ;
x=8 ;
y=x ;
```

VI. Constantes

Une *constante* est un identificateur qui contient une valeur qui ne sera jamais modifiée au cours du programme. Par exemple, la constante PI peut être fixée à 3.14.
Les constantes peuvent être des nombres entiers (12), des nombres réels (20.6), un caractère ('a') ou une chaîne de caractères ("bonjour"). On les écrit souvent en majuscules pour les différencier des autres identificateurs.

Déclaration de constante

Il existe deux façons pour déclarer une constante :

Syntaxe 1 :

```
#definenom_constantevaleur_constante
```

Exemples :

```
#define  PI 3.14
#define TVA 0.16
```

Cette déclaration est effectuée dans la partie préprocesseurs (après l'inclusion des bibliothèques) et elle permet la déclaration d'une constante qui sera visible partout le programme et sa durée de vie est celle de toute l'application.

Syntaxe 2 :

```
const type nom_constante = valeur_constante ;
```

Exemple :

```
const float Pi = 3.14 ;
```

Cette déclaration est effectuée dans le bloc et elle permet la déclaration d'une constante qui sera visible seulement dans le bloc et elle sera détruite en sortant du bloc.

VII. L'affichage et la saisie des variables

VII.1. La fonction d'affichage printf()

C'est une fonction implémentée dans la bibliothèque standard **stdio.h**, elle permet d'afficher des données à l'écran.

Syntaxe :

printf ("String Format" [, argument$_1$…argument$_n$]);

- String format : chaîne de caractères avec des caractères de formatage si nécessaire
- Un argument peut être : une valeur ou une variable ou une expression ou un appel de fonction.

Le nombre des arguments après le String Format est variable et dépend du nombre de variables pour lesquels vous voulez afficher les valeurs. Sachez que pour afficher une variable, encore une fois, cela dépend de son type, effectivement, même si une fois afficher, un nombre ou une chaine de caractère ne fait qu'apparaître bêtement à l'écran, la façon dont le système d'exploitation procède pour vous l'afficher diffère selon le type de la variable. Il existe beaucoup de format. Le tableau suivant montre les formats les plus utilisés.

format	signification
%c	Caractère
%s	chaîne de caractères
%d	nombre entier en décimal
%e	nombre réel sous la forme mantisse/exposant [-]m.nnnnnne[+|-]xx
%E	nombre réel sous la forme mantisse/exposant en majuscule [-]m.nnnnnnE[+|-]xx
%f	nombre réel sous la forme [-]mmm.nnnnnn

format	signification
%g	nombre réel sous la forme la plus courte entre les types %e et %f
%G	nombre réel sous la forme la plus courte entre les types %E et %f
%o	nombre entier en octal
%p	pointeur ou adresse de la valeur numérique
%u	nombre entier non signé en décimal
%x	nombre entier en hexadécimal
%X	nombre entier en hexadécimal ; lettres affichées en majuscules

Exemple:

Dans l'exemple suivant nous avons utilisé la fonction **printf** qui permet d'afficher un message texte à l'écran. **printf** n'est pas une instruction du langage C, mais une fonction de la bibliothèque **stdio.h.**

```
#include  <stdio.h>
#include <conio.h>
void main ( ){
        int x=8 ;
        float y=5.6 ;
        clrscr ( ) ;
        printf(" Bonjour") ;
        printf("x= %d et y=%f", x, y) ;

printf("\n pour continuer frapper une touche") ;
/* \n permet le retour à la ligne*/
getch  ( ) ;
}
```

VII.2. La fonction de saisie scanf()

C'est une fonction implémentée dans la bibliothèque standard **stdio.h,** elle permet la saisie des variables : elle modifie leurs contenus.

Syntaxe :

scanf ("Format1 […Format$_n$]", &argument$_1$[, … &argument$_n$]);

- Format doit être un format du tableau précédent (%d, %f…)
- Argument ne peut être qu'une variable dont la valeur va être saisie par l'utilisateur
- & est obligatoire sauf pour la saisie des variables de type pointeur

Exemple:

```
int x;
float y;
scanf ("%d %f" , &x, &y);
```

VIII. Travail demandé

Exercice 1 :
Ecrire un programme qui saisit deux caractères A et B et qui réalise une permutation de ces deux variables.

Exercice 2 :
Ecrire un programme qui permet de calculer la surface d'un disque de rayon R.

Exercice 3 :
Écrire un programme qui permet de saisir un entier n et d'afficher séparément chacun de ces trois chiffres : Unité. Dizaine et centaine. On suppose que le nombre est inférieur à 1000.

Exercice 4 : E/S conversationnelles
Quels résultats fournit le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
void main ( ) {
        int n=543;
        int p=5;
        float x=34.5678;
        printf ( "A : %d %f \n" ,n , x ) ;
        printf ( "B : %4d %10f \n" ,n , x ) ;
        printf ( "C : %2d %3f \n" ,n , x ) ;
        printf ( "D : %10.3 f  %10.3 e \n" ,x , x ) ;
        printf ( "E : %-5d  %f \n" ,n , x ) ;
        printf ( "F : %*d\n" ,p , n ) ;
        printf ( "G : %*.*f \n" , 1 2 , 5 , x ) ;
        printf ( "H : %x : %8x : \n" ,n , n ) ;
        printf ( " I : %o : %8o : \n" ,n , n ) ;
}
```

Exercice 5 : E/S conversationnelles
1) Quelles seront les valeurs des deux variables n et p (de type int), par l'instruction suivante :

```
scanf ( "%d %d",&n, &p ) ;
```

Lorsqu'on lui fournit les données suivantes (le symbole ^ représente un espace et le symbole @ représente une fin de ligne, c'est-à-dire une "validation").

(a) 253^45@
(b) ^253^@
(c) ^^4^5@

2) Quelles seront les valeurs des deux variables n et p (de type int), par l'instruction suivante :

```
scanf ( "%4d %2d",&n,&p ) ;
```

Lorsqu'on lui fournit les données suivantes :

(a) 12^45@
(b) 123456@
(c) 123456^7@
(d) 1^458@
(e) ^^^4567^^8912@