



T.P. 2

Initiation au langage C

DEA SyAM 2000/2001

Par
Philippe Lucidarme : lucidarm@lirmm.fr

Initiation au langage C

Les fonctions

On appelle fonction un sous-programme qui permet d'effectuer un ensemble d'instructions par simple appel de la fonction dans le corps du programme principal. Les fonctions permettent d'exécuter dans plusieurs parties du programme une série d'instructions, cela permet une simplicité du code et donc une taille de programme minimale. D'autre part, une fonction peut faire appel à elle-même, on parle alors de fonction récursive (il ne faut pas oublier de mettre une condition de sortie au risque sinon de ne pas pouvoir arrêter le programme...).

Avant d'être utilisée, une fonction doit être définie car pour l'appeler dans le corps du programme il faut que le compilateur la connaisse, c'est-à-dire qu'il connaisse son nom, ses arguments et les instructions qu'elle contient. La définition d'une fonction s'appelle "déclaration". La déclaration d'une fonction se fait selon la syntaxe suivante:

```
type_de_donnee Nom_De_La_Fonction (type1 argument1, type2 argument2, ...)
{
    liste d'instructions
}
```

Tapez et exécutez le programme suivant :

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

void afficher_bonjour ()           // Déclaration de la fonction
{
    printf ("Bonjour !");
}

void main ()                       // Programme principal
{
    afficher_bonjour ();
    getch ();
}
```

Dans le cas de `afficher_bonjour`, nous ne retournons aucun paramètre : la déclaration commence par un `void`. Voici un exemple de fonction qui permet de saisir un chiffre et le renvoie au programme principal (fonction `return`).

```
int lire_chiffre ()               // Déclaration de la fonction
{
    int a;
    printf ("Entrez un nombre : ");
    scanf ("%d",&a);
    return (a);                  // La valeur retournée sera a
}
```

Dans le programme principal, la valeur lire_chiffre pourra être utiliser comme une variable :

```
int chiffre1=lire_chiffre ();
```

Ecrire une fonction qui permet à l'utilisateur de saisir un chiffre entier, et renvoie un 1 si ce chiffre est pair et un 0 s'il est impair. Le programme principal permettra de demander 5 chiffres à l'utilisateur et affichera le nombre de chiffres pairs.

Nous pourrons utiliser l'opérateur % (modulo). $a \% b$ = reste de la division entière de a par b.

Les fonctions paramétrées

Nous venons de voir qu'il été possible de passer des valeurs de la fonction vers le programme principal, mais l'inverse est également possible :

Saisissez le programme suivant :

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

float robot (    float diam,
                 float nb_tours)          // Déclaration de la fonction
{
    float tmp;
    tmp = nb_tours * 3.14 * diam;
    return (tmp);
}

void main ()    // Programme principal
{
    float D,Nb;
    printf ("Entrez le diametre des roues : ");
    scanf ("%f",&D);
    printf ("Entrez le nombre de tours de roues : ");
    scanf ("%f",&Nb);
    printf ("Le robot a avance de %2f cm",robot(D,Nb) );
    getch ();
}
```

La fonction robot renvoie la distance parcourue par un robot, connaissant le diamètre de ses roues et le nombre de tours.

Ecrire une fonction paramétrée qui permet de calculer le déterminant de l'équation du second degré suivante :

$$a.x^2 + b.x + c = 0$$

On pourra passer en paramètre les valeurs de a, b et c.

Ecrire trois fonctions qui permettent d'afficher les racines solutions de l'équation précédente.

1. Dans le cas où l'on a deux racines réelles.
2. Dans le cas où l'on a une racine double.
3. Dans le cas où l'on a deux racines complexes.

L'on pourra passer en paramètre les valeurs de a, b et c.

Enfin, écrire le programme principal qui permet à l'utilisateur de saisir a, b et c. Celui-ci calcule la valeur du déterminant, puis appelle une des trois fonctions ci-dessus selon le signe du déterminant.

Les tableaux

Les variables, telles que nous les avons vues, ne permettent de stocker qu'une seule donnée à la fois. Or, pour de nombreuses données, comme cela est souvent le cas, des variables distinctes seraient beaucoup trop lourdes à gérer. Heureusement, le langage C propose des structures de données permettant de stocker l'ensemble de ces données dans une "variable commune". On appelle tableau une variable composée de données de même type, stockée de manière contiguë en mémoire (les unes à la suite des autres).

Déclaration d'un tableau :

Type_de_donnée Nom_du_tableau [Nombre d'éléments] ;

Exemple : int Tab [10] ;

Le programme crée la structure suivante en mémoire :

int	int	int	int	int	int	int	int	int	int
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

On accède à un élément de la façon suivante :

Nom_du_tableau [Numéro de l'élément]

Attention, lorsque vous créez un tableau, celui-ci commence à l'indice 0. Si vous créez un tableau de n éléments, le premier sera 0 et le dernier n-1.

Saisissez le programme suivant :

```
#include <stdio.h>
#include <conio.h>
void main ()
{
    int tab [10];
    for (int i=0;i<10;i++)
    {
        printf ("Entrez le nombre N %d : ",i+1);
        scanf ("%d",&tab [i]);
    }
    for (i=0;i<10;i++)
    {
        printf ("Tab (%d)=%d \n",i,tab[i]);
    }
    getch ();
}
```

Modifiez le programme pour qu'il affiche uniquement le plus grand des dix chiffres saisis.

Les tableaux multidimensionnels

Nous venons de voir qu'il était possible de créer des tableaux de données, mais il est également possible de créer des tableaux de tableaux. Autrement dit, des tableaux à plusieurs dimensions.

Déclaration d'un tableau multidimensionnel:

Type_de_donnée Nom_du_tableau [Nombre d'éléments 1] [Nombre d'éléments 2] ;

Exemple : int Tab [4] [3] ;

Le programme crée la structure suivante en mémoire :

Tab [0] [0]	Tab [1] [0]	Tab [2] [0]	Tab [3] [0]
Tab [0] [1]	Tab [1] [1]	Tab [2] [1]	Tab [3] [1]
Tab [0] [2]	Tab [1] [2]	Tab [2] [2]	Tab [3] [2]

On accède à un élément de la façon suivante :

Nom_du_tableau [Numéro de l'élément 1] [Numéro de l'élément 2]

Ecrire un programme qui réalise la somme de deux matrices carrées de dimension 2.

On pourra initialiser les matrices de la façon suivante :

```
int mat1 [2] [2] = {1,2,3,4};
int mat2 [2] [2] = {5,6,7,8};
```

Modifier le programme pour faire la somme de deux matrices carrées de dimensions 3.

Les pointeurs

Un des avantages du C, est de pouvoir utiliser la mémoire comme bon nous semble : il est en effet possible de connaître l'adresse à laquelle une variable est stockée en mémoire. Un pointeur est une variable contenant l'adresse d'une autre variable d'un type donné.

Prenons l'exemple suivant :

Nous déclarons une variable :

```
char a = 'a' ;
```

Le programme va allouer un espace mémoire pour cette variable a.

Adresse mémoire	8541	8542	8543	8544	8545	8546	8547	8548
Contenu	X	x	x	61	x	x	x	x

Dans notre exemple, l'emplacement mémoire est le 8544, on y voit stocké la valeur 61, qui correspond à la valeur hexadécimale du code ASCII de 'a'

Nous allons maintenant allouer une adresse mémoire pour y stocker l'adresse de la variable a :

```
Int *p1 ;
```

p1 est un pointeur, c'est l'étoile qui fait la différence avec une variable normale.

Nous allons maintenant stocker l'adresse de a dans p1 :

```
p1 = &a ;
```

Le caractère & placé avant une variable permet de renvoyer l'adresse mémoire de celle-ci.

Le contenu de p1 sera 8544

Saisissez le programme suivant :

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

```
void main ()
{
char a='a';
char *p1;
p1=&a;
printf ("le contenu de la variable a est %d.\nIl se trouve a l'adresse memoire %p",*p1,p1);
getch ();
}
```

Remarque : L'adresse mémoire est constituée de deux parties : xxxx yyyy. Où xxxx est le segment de mémoire, et yyyy l'offset.

Ecrire un nouveau programme :

Déclarer le tableau suivant

```
int tab [7]={84,104,101,32,101,110,100};
```

Lire l'adresse mémoire de chacun des éléments. Afficher le contenu de chaque adresse sous forme de caractère.