Chapitre 7

Les fonctions

La programmation modulaire

- Certains problèmes conduisent à des programmes longs, difficiles à écrire et à comprendre. On les découpe en des parties appelées **sous-programmes** ou **modules**
- Les modules sont des groupes d'instructions qui fournissent une solution à des parties bien définies d'un problème plus complexe. Ils ont plusieurs **intérêts** :
  - permettent de **"factoriser" les programmes**, càd de mettre en commun les parties qui se répètent
  - permettent une **structuration** et une **meilleure lisibilité** des programmes
  - **facilitent la maintenance** du code (il suffit de modifier une seule fois)
  - peuvent éventuellement être **réutilisées** dans d'autres programmes
- La structuration de programmes en sous-programmes se fait en C à l'aide des **fonctions**

Fonctions

- On définit une fonction en dehors de la fonction principale main ( ) par :

  type nom_fonction (**type1** arg1,…, **typeN** argN)

  {

  instructions constituant le corps de la fonction

  return (expression)

  }

- Dans la première ligne (appelée **en-tête de la fonction**) :
  - **type** est le type du résultat retourné. Si la fonction n'a pas de résultat à retourner, elle est de type **void.**
  - le choix d'un nom de fonction doit respecter les mêmes règles que celles adoptées pour les noms de variables.
  - entre parenthèses, on spécifie les **arguments** de la fonction et leurs types. Si une fonction n'a pas de paramètres, on peut déclarer la liste des paramètres comme **(void)** ou simplement comme **()**
- Pour fournir un résultat en quittant une fonction, on dispose de la commande **return.**

Fonctions : exemples

- Une fonction qui calcule la somme de deux réels x et y :

```
double Som(double x, double y )
        {
          return (x+y);
        }
```

- Une fonction qui affiche la somme de deux réels x et y :

```
void AfficheSom(double x, double y)
      {
         printf (" %lf", x+y );
      }
```

- Une fonction qui renvoie un entier saisi au clavier

```
int RenvoieEntier( void )
      {
        int n;
        printf (" Entrez n \n");
       scanf (" %d ", &n);
       return n;
      }
```

- Une fonction qui affiche les éléments d'un tableau d'entiers

```
void AfficheTab(int T[ ], int n)
      {  int i;
         for(i=0;i<n;i++)
         printf (" %d \t", T[i]);
      }
```

Appel d'une fonction

- L'appel d'une fonction se fait par simple écriture de son nom avec la liste des paramètres : nom_fonction (para1,…, paraN)
- Lors de l'appel d'une fonction, les paramètres sont appelés **paramètres effectifs** : ils contiennent les valeurs pour effectuer le traitement. Lors de la définition, les paramètres sont appelés **paramètres formels**.
- L'ordre et les types des paramètres effectifs doivent correspondre à ceux des paramètres formels
- **Exemple d'appels:**

```
main( )
    {   double z;
        int A[5]   = {1, 2, 3, 4, 5};
       z=Som(2.5, 7.3);
       AfficheTab(A,5);
    }
```

Déclaration des fonctions

- Il est nécessaire pour le compilateur de connaître la définition d'une fonction au moment où elle est appelée. Si une fonction est définie après son premier appel (en particulier si elle définie après main), elle doit être **déclarée** auparavant.

- La déclaration d'une fonction se fait par son **prototype** qui indique les types de ses paramètres et celui de la fonction :

  type nom_fonction (**type1**,…, **typeN**)

- Il est interdit en C de définir des fonctions à l'intérieur d'autres fonctions. En particulier, on doit définir les fonctions soit avant, soit après la fonction principale main.

Déclaration des fonctions : exemple

```
#include<stdio.h>
float ValeurAbsolue(float); //prototype de la fonction ValeurAbsolue
main( )
 { float  x=-5.7,y;
   y= ValeurAbsolue(x);
    printf("La valeur absolue de %f est : %f \n " , x,y);
 }
//Définition de la fonction  ValeurAbsolue
 float ValeurAbsolue(float a)
   {
      if (a<0) a=-a;
      return a;
   }
```

Variables locales et globales

- On peut manipuler 2 types de variables dans un programme C : des **variables locales** et des **variables globales**. Elles se distinguent par ce qu'on appelle leur **portée** (leur "espace de visibilité", leur "durée de vie")

- Une variable définie à l'intérieur d'une fonction est une **variable locale**, elle n'est connue qu'à l'intérieur de cette fonction. Elle est créée à l'appel de la fonction et détruite à la fin de son exécution

- Une variable définie à l'extérieur des fonctions est une **variable globale**. Elle est définie durant toute l'application et peut être utilisée et modifiée par les différentes fonctions du programme.

Variables locales et globales : remarques

- Les variables déclarées au début de la fonction principale main ne sont pas des variables globales, mais elles sont locales à main
- Une variable locale cache la variable globale qui a le même nom
- Il faut utiliser autant que possible des variables locales. Ceci permet d'économiser la mémoire et d'assurer l'indépendance de la fonction
- En C, une variable déclarée dans un bloc d'instructions est uniquement visible à l'intérieur de ce bloc. C'est une variable locale à ce bloc, elle cache toutes les variables du même nom des blocs qui l'entourent

Variables locales et globales : exemple

```
#include<stdio.h>
int x = 7;
int f(int);
int g(int);
main( )
{   printf("x = %d\t", x);
    {  int x = 6;   printf("x = %d\t", x); }
    printf("f(%d) = %d\t", x, f(x));
    printf("g(%d) = %d\t", x, g(x));
 }
int f(int a)  { int x = 9;   return (a + x); }
int g(int a) { return (a * x); }
```

Qu'affiche ce programme?

x=7 x=6 f(7)=16 g(7) = 49

Variables locales et globales : exemple

```
#include<stdio.h>
void f(void);
int i;
main( )
{ int k = 5;
  i=3; f(); f();
 printf("i = %d et k=%d \n", i,k); }
void f(void)  { int k = 1;
                printf("i = %d et k=%d \n", i,k);
                i++;k++;}
```

Qu'affiche ce programme?

i=3 et k=1
i=4 et k=1
i=5 et k=5

Paramètres d'une fonction

- Les paramètres servent à échanger des informations entre la fonction appelante et la fonction appelée. Ils peuvent recevoir des données et stocker des résultats

- Il existe deux modes de transmission de paramètres dans les langages de programmation :

  - **La transmission par valeur** : les valeurs des paramètres effectifs sont affectées aux paramètres formels correspondants au moment de l'appel de la fonction ou procédure. *Dans ce mode le paramètre effectif ne subit aucune modification*

  - **La transmission par adresse (ou par référence)** : les adresses des paramètres effectifs sont transmises à la fonction appelante. *Dans ce mode, le paramètre effectif subit les mêmes modifications que le paramètre formel*

Transmission des paramètres en C

- La transmission des paramètres en C se fait toujours par valeur
- Pour effectuer une transmission par adresse en C, on déclare le paramètre formel de type pointeur et lors d'un appel de la fonction, on envoie l'adresse et non la valeur du paramètre effectif
- Exemple :

```
void Increment (int x,  int *y)
        { x=x+1;
         *y =*y+1; }
main( )
        { int n = 3, m=3;
          Increment (n, &m);
          printf("n = %d et m=%d \n", n,m);  }
```

Résultat :

n=3 et m= 4

Exemples

Une fonction qui échange le contenu de deux variables :

```
void Echange (float *x,  float *y)
    { float z;
            z = *x;
           *x = *y;
           *y = z;
   }
main()
    { float a=2,b=5;
      Echange(&a,&b);
      printf("a=%f,b=%f\n ",a,b);
   }
```

Récursivité

- Une fonction qui fait appel à elle-même est une fonction **récursive**
- Toute fonction récursive doit posséder un cas limite (cas trivial) qui arrête la récursivité
- Exemple : Calcul du factorielle

```
int  fact (int n )
    {     if (n==0)    /*cas trivial*/
                  return (1);
          else
                   return (n* fact(n-1) );
    }
```

Remarque : l'ordre de calcul est l'ordre inverse de l'appel de la fonction

Fonctions récursives : exercice

- Ecrivez une fonction récursive (puis itérative) qui calcule le terme n de la suite de Fibonacci définie par : U(0)=U(1)=1
  U(n)=U(n-1)+U(n-2)

```
int  Fib (int n)
   {
        if (n==0 || n==1)
              return (1);
        else
              return ( Fib(n-1)+Fib(n-2));

   }
```

Fonctions récursives : exercice (suite)

- Une fonction itérative pour le calcul de la suite de Fibonacci :

```
int  Fib (int n)
   { int i, AvantDernier, Dernier, Nouveau;
        if (n==0 || n==1) return (1);
        AvantDernier=1; Dernier =1;
         for (i=2; i<=n; i++)
            { Nouveau= Dernier+ AvantDernier;
              AvantDernier = Dernier;
              Dernier = Nouveau;
            }
        return (Nouveau);
   }
```

Remarque: la solution récursive est plus facile à écrire