CH4
Les sous programmes

1ère Année Licence GL

Kais ben Salah
&
Taoufik Sakka Rouis

Les sous programmes

1- Introduction

- Dans la résolution d'un problème, on peut constater qu'une suite d'actions revient plusieurs fois.
- Dans ce cas il serait judicieux de l'écrire une seule fois, et de l'utiliser autant de fois que c'est nécessaire, en effectuant des calculs avec des données différentes.
- Cette suite d'actions sera définie dans un sous programme, qui peut prendre soit la forme d'une fonction.
- Une fonction a toujours un type de retour, qui correspond au type du résultat qu'elle peut renvoyer et qui peut être n'importe lequel des types que nous avons précédemment étudié.

Les sous programmes

- Le type de retour peut être **void** si on souhaite que la fonction ne renvoie rien.
- Une fonction a aussi un nom et une liste de **zéro**, **un**, ou **plusieurs**, paramètres.
- D'autre part, on peut observer que certains groupes d'actions se rapportent à des traitements précis et différents. Il est souhaitable alors de représenter chacun d'eux dans un sous programme.

→ On perçoit alors un programme comme un ensemble de fonctions.

Les sous programmes

2- Avantages d'utilisation des fonctions

- D'améliorer la lisibilité des programmes.
- Optimiser le nombre d'instructions dans un programme.
- De faciliter la mise au point et la correction des erreurs.

Les sous programmes

3- Définition d'une fonction

Une fonction est un sous programme qui peut être appelée par la fonction principale du programme (main) ou par une autre fonction.

Une définition de fonction spécifie :

- Le nom de la fonction,
- Le type, le nombre et les noms des paramètres de la fonction,
- Le type du résultat fourni par la fonction,
- Les données locales à la fonction,
- D'autres fonctions appelées par la fonction,
- Les instructions à exécuter.

Les sous programmes

Syntaxe :

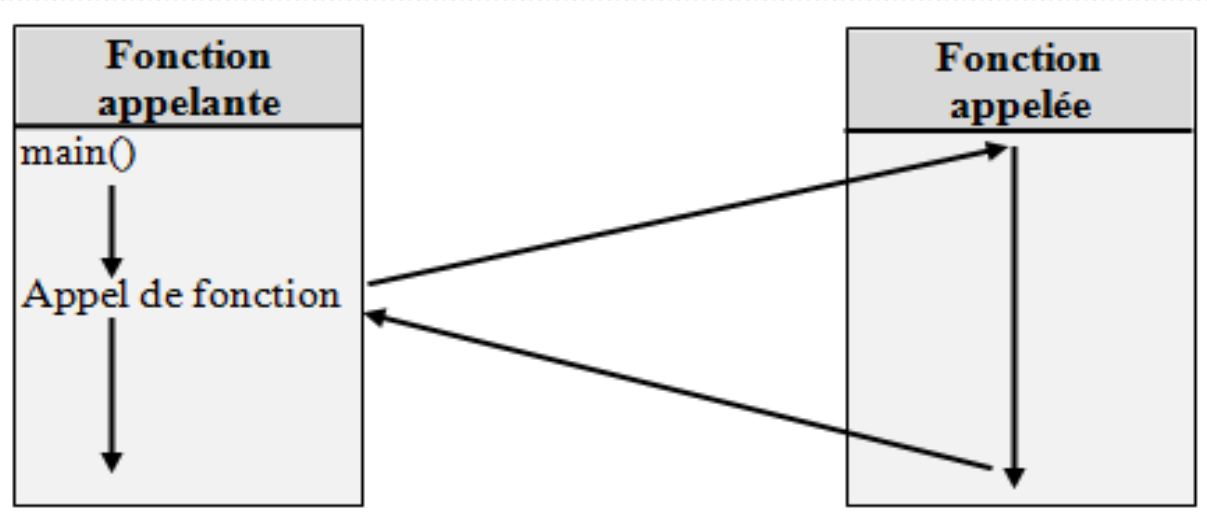
```
<typeRés><NomFonct>(TypePar1><NomPar1>,<TypePar2><NomPar2>,...)
{
    <déclarations locales> ;
    <instructions> ;
    return Résultat ; /*optionnelle si la fonction ne  retourne rien
                      /*(de type void)*/
}
```

Les sous programmes

4- Appel d'une fonction

On appel une fonction en mentionnant son nom, suivi d'une liste d'arguments entre parenthèses et séparés par des virgules.

Schéma d'appel



Les sous programmes

5. Fonction fournissant un résultat

5.1 Fonction fournissant un résultat et avec paramètres

Exemple :

```
# include<stdio.h>
/* Définition de la fonction som */
int som( int u , int v)
{   int s ;
    s = u + v ;
    return s ;
}
void main()
{ int a=1, b=2, c=3, d=4, x, y  ; /*Déclaration des variables*/
  x = som (a,b) + 5; /*Appel de la fonction avec les arguments a et b*/
  printf ("x = %d \t" , x) ;
  y = 3 * som (c,d) ;
  printf ("y = %d \n" , y) ;
}
```

Les sous programmes

5.2. Quelques règles :

a) Arguments Muets , Arguments Effectifs :

- Les noms des arguments figurant dans l'en-tête de la fonction se nomment des « **arguments muets** » (ou encore "**arguments formels**" ou "**paramètres formels** ").

  → Leur rôle est de permettre, au sein du corps de la fonction, de décrire ce qu'elle doit faire.

Exemple : u et v sont des paramètres formels

- Les arguments fournis lors de l'utilisation (l'appel) de la fonction se nomment des « **arguments effectifs** » (ou encore "**paramètres effectifs**").

Exemple : a, b, c et d sont des paramètres effectifs

Les sous programmes

b) Lors de la définition

L'entête de la fonction précise toujours la liste des arguments formels avec leurs types.

c) Lors de l'appel

L'utilisation de la fonction « som » au sein du programme C se fait toujours en faisant suivre son nom et d'une liste d'arguments effectifs.

→ Celle-ci précise au compilateur que « som » est une fonction et qu'elle fournit un résultat de type « double ».

Les sous programmes

d) L'instruction «return»

- « **return** » retourne le résultat et interrompt l'exécution de la fonction qui la appelée.
- L'instruction « **return** » peut mentionner n'importe quelle expression.

Exemple :

```
int  som( int  u , int v)
{
      return (u+v) ;
}
```

Les sous programmes

- L'instruction «**return**» peut apparaître à plusieurs reprises dans une fonction.

```
double absom (double u , double v)
{
   double s;
   s = u + v;
   if (s>0)
        return(s);
   else
        return ( -s);
}
```

Les sous programmes

5.3 Fonction fournissant un résultat et sans paramètres

Exemple :

```
# include<stdio.h>
int entree()
{  int nombre;
   printf("Donner un entier :");
   scanf("%d", &nombre);
   return nombre;
}
void main()
{    int A,B;
     A=entree();
     B=entree();
     printf("La somme =%d", A+B);
}
```

Résultat :

Donner un entier : 3
Donner un entier : 5
La somme = 8

Les sous programmes

6. Cas des fonctions sans valeur de retour (type void)

6.1. Cas d'une fonction avec arguments et sans valeur de retour

Exemple :

```
#include<stdio.h>
void ecris(int n)
{
      printf("Valeur :%d\n",n);
}
void main()
{
      int a=10,b=20;
      ecris(a);
      ecris(b);
      ecris(a+b);
}
```

Résultat :

Valeur : 10
Valeur : 20
Valeur : 30

Les sous programmes

6.2. Cas d'une fonction sans arguments et sans valeur de retour

Exemple :

```
# include<stdio.h>
void optimist()
{
      printf("Il fait beau");
}
void main()
{
      optimist();
}
```

Résultat :
Il fait beau

Les sous programmes

7 - Paramètres d'une fonction

on distingue **2 méthodes** pour passer des paramètres effectifs à une fonction :

7.1 Passage des paramètres par valeur

Exemple :

```
#include<stdio.h>
void PERMUTER (int X, int Y)
{   int AIDE;
    AIDE = X;
    X =Y;
    Y = AIDE;
}
void main()
{
     int X=3,Y=4;
     printf("Avant appel de PERMUTER : X=%d\t Y=%d\n", X,Y);
     PERMUTER (X,Y);
     printf("Après appel de PERMUTER : X=%d\t Y=%d", X,Y);
}
```

Les sous programmes

Exécution :

Avant appel de PERMUTER : X = 3 Y = 4

Après appel de PERMUTER : X = 3 Y = 4

⟶ **X et Y restent échangés.**

- Lors de l'appel, les valeurs de X et Y sont copiées dans les paramètres A et B. PERMUTER échange bien le contenu des variables locales A et B, mais les valeurs de X et Y restent les mêmes.

- Pour changer la valeur d'une variable de la fonction appelante, nous allons procéder comme suit :
  - ✓ La fonction appelante doit fournir **l'adresse** de la variable et
  - ✓ La fonction appelée doit déclarer le paramètre comme **pointeur**.

Les sous programmes

7.2 Passage des paramètres par adresse

- Dans ce cas la fonction ne fait pas de copie de la valeur du paramètre, comme c'est fait dans le cas du passage par valeur.
- Par conséquent, si le paramètre change de valeur, alors à la sortie de la fonction, la variable passée en paramètre contient la dernière modification, donc la dernière valeur.

Exemple :

```
#include<stdio.h>
void PERMUTER (int *X, int *Y)
{    int AIDE;
     AIDE = *X;
     *X = *Y;
     *Y = AIDE;
}
void main()
{    int X=3, Y=4;
     printf("Avant appel de PERMUTER : X=%d\t Y=%d\n ", X, Y);
     PERMUTER (&X, &Y);
     printf("Après appel de PERMUTER : X=%d\t Y=%d\n ", X, Y);
}
```

Les sous programmes

Exécution :

Avant appel de PERMUTER : X = 3 Y = 4

Après appel de PERMUTER : X = 4 Y = 3

⟶ Lors de l'appel, les adresse de X et Y sont copiées dans les pointeurs A et B. PERMUTER échange ensuite le contenu des adresses indiquées par les pointeurs A et B.

Les sous programmes

8. Variables globales et variables locales

- En C, plusieurs fonctions (dont, bien entendu le programme principal main) peuvent partager des variables communes qu'on qualifie alors de globales.

- Les variables globales sont déclarées immédiatement derrière les instructions **#include** au début du programme.

- Les variables locales ne sont connues qu'à l'intérieur de la fonction où elles sont déclarées : leur portée est donc limitée à cette fonction.

Les sous programmes

Exemple :

```
int n ; /*Variable globale*/
fct1()
{
    int p ; /*Variables locales dans fct1*/
.........
}
void main()
{
    int p ; /*Variable locale dans main*/
    .....
}
```

- La variable p de main n'a aucun rapport avec la variable p de fct1.
- La variable n est une variable globale, elle peut être utilisée soit dans « fct1 » ou « main ».

Les sous programmes

9. Les prototypes

- Le prototype d'une fonction est une description d'une fonction qui est définie plus loin dans le programme. On place donc le prototype en début du programme (avant la fonction principale main()).

- Le prototype d'une fonction reprend exactement l'en-tête de la fonction, mais pas son corps, qui est remplacé par **un point-virgule**.

- Cette description permet au compilateur de « vérifier » la validité de la fonction à chaque fois qu'il la rencontre dans le programme.

Les sous programmes

Syntaxe :

```
TypeDeRetour  NomFN (Type1 Parametre1, Type2 Parametre2) ;
```

Exemple :

```
# include <stdio.h>
int somme (int a, int b); /*déclaration du prototype de la fonction somme */
void main ()
 {
      int x, y, z;
      ...
      z= somme (x, y) ;
       ...
}
/* implémentation du corps de la fonction somme */
int somme (int a, int b)
 {
                ...
}
```