C

Ch 5 – Fonctions

I. Introduction

Le langage C définit une seule forme de sous-programme : la **fonction**.
Cependant le langage met en œuvre un type de retour particulier (**void**) pour distinguer les fonctions qui retournent une valeur (« expressions ») de celles qui ne retournent aucune valeur (« instructions »)

Vous avez déjà utilisé une fonction dans les programmes écrits en C : en effet, le point d'entrée d'un programme C est la **fonction** « **main** » , qui renvoie une valeur de type « int ».

Exemple 1 : (sans paramètres)

```
int main ()
{
. . .// déclarations et instructions
  return 0 ; // retourne la valeur 0 à l'appelant (le système)
}
```

Exemple 2 : (avec les paramètres standard de la fonction main)

```
int main (int argc, char *argv[])
{
. . .// déclarations et instructions
```

```
  return 0 ; // retourne la valeur 0 à l'appelant (le système)
}
```

II. Fonctions (1) – sous-programme « expression »

Une **FONCTION** C est un **SOUS-PROGRAMME** qui
- possède un **TYPE DE DONNEES DE RETOUR AUTRE QUE `void`**
- inclut l'**INSTRUCTION `return`** qui **VA RENVOYER UNE VALEUR RESULTAT** au programme appelant**.**

A. DEFINIR une fonction : ENTETE et CORPS de la fonction

Syntaxe de la DEFINITION d'une fonction:

```
type_retour nom_fonction (par1,...,parN)
{
  // corps de la fonction : déclaration et instructions
  . . .
  return valeur_retour ;
}
```

- **type_retour :** type de données du résultat renvoyé par l'instruction **return**
  - type de données de base : int, double, char, string, bool,
  - structures (mot clef struct)
- **nom_fonction** : identifiant de la fonction (exemple de convention : préfixé par 'f')
- **par1, . . ., parN** : déclaration des paramètres attendus par la fonction
- **return = QUITTE LA FONCTION ET RENVOIE LA VALEUR**
  - **valeur_retour :** valeur littérale ou expression ; même type que **type_retour**

Exemple :
Définition de la fonction fCalcDuree :

```
int fCalculerDuree (int pAnDeb, int pAnFin )
{
   return pAnFin – pAnDeb  ;
}
```

Utilisation de la fonction fCalculerDurée :

```
. . .
   int anNais, anCour ;
   scanf("%d",&anNais);
   scanf("%d",&anCour);
   printf("%d ans", fCalculerDuree(anNais,anCour));
 . . .
```

III.Fonctions (2) – sous-programme « instruction »

Une **PROCEDURE** C est un **SOUS-PROGRAMME** qui
- possède un **TYPE DE DONNEES DE RETOUR `void`**
- et qui ne renvoie de valeur de retour

mais qui **peut utiliser l'instruction `return`** pour quitter et retourner à l'instruction qui suit le point d'appel.

A. DEFINIR une fonction : ENTETE et CORPS de la fonction

Syntaxe:

```
void nom_procedure (par1,...,parN)
{
  // Corps de la procédure : déclaration et instructions
  . . .
  [return ;]
}
```

- **nom_procedure** : nom identifiant la procédure (exemple de convention : préfixé par 'p')
- **par1, . . ., parN** : déclaration des paramètres attendus par la procédure

Exemple de définition sans déclaration de paramètres :
Définition de la procédure pAfficherMenu (aucun paramètre)

```
void pAfficherMenu ()
{
    printf( "Menu");
    printf("====");
    printf("choix 1 : surface d'un rectangle");
    printf("choix 2 : surface d'un cercle");
    printf("choix 0 : quitter");
}
```

Utilisation de la procédure :pAfficherMenu :

```
. . .
    int choix ;
    pAfficherMenu() ;
    scanf("%d", &choix ;
    . . .
```

Exemple de définition avec déclaration de paramètres :
Définition de la procédure pAffSomme :

```
void pAfficherSomme(int pNb1, int pNb2 )
{
    printf( "la somme est %d" ,pNb1 + pNb2 );
}
```

Utilisation de la procédure :pAffSomme :

```
    int n1, n2 ;
    scanf("%d",&n1);
    scanf("%d",&n1);
    pAfficherSomme(n1, n2) ;
    . . .
```

IV. Paramètres (Arguments)

A. Mode de passage des paramètres

1. Passage par valeur

Dans le mode de **PASSAGE PAR VALEUR en C**, chaque paramètre est déclaré comme n'importe quelle variable (ou constante) et, au moment de l'appel, il est affectée de la valeur de l'argument fourni lors de l'appel.

Exemple :

```
// Définition fonc.
int fCalculerSurfCarre(int pCote)
{
    return (pCote * pCote); // retourner la surface
}

int main ()
{
    int cote;

    printf( "Entrez le coté :");
    scanf("%d",&cote);
    printf( "la surface est %d" , fCalcSurfCarre(cote) )
. . .
    return EXIT_SUCCESS; // constante fournie par C, vaut 0
}
```

2. Passage par référence : pointeurs

Le **PASSAGE PAR REFERENCE en C est réalisé en utilisant des pointeurs dans la définition des paramètres.**

(cf. cours sur les pointeurs)

Cf. Annexe sur le passage des tableaux en paramètres.

V. DECLARATION : prototype de fonction

TOUTE FONCTION C DOIT ETRE CONNUE AVANT SON APPEL : ELLE DOIT DONC :

- **SOIT ETRE DEFINIE AVANT SON UTILISATION** dans le programme source (implémentation complète : entête et corps – cf. Fonctions et Procédures
- **SOIT ETRE DECLAREE AVANT SON UTILISATION** (déclaration d'une forme réduite de la fonction, le PROTOTYPE, la définition complète étant réalisée à un autre endroit du programme source ou dans un autre programme source – cf. DECLARATION : prototype de fonction et Organisation des fichiers d'un programme C)

A. Prototype

Le prototype d'une fonction correspond à la partie de l'entête qui va permettre au compilateur d'identifier cette fonction, de connaître les types des paramètres (éventuellement leur valeur par défaut)

Syntaxe :

```
type_retour nom_fonction (typ1,...,typN) ;
```

- **nom_fonction** est le nom donné à la fonction,

- **typ1, typ2, typN** : mode de passage et type de données des arguments attendus par la fonction, avec possibilité de spécifier de valeurs par défaut
- **type_retour** : type de la donnée retournée par la fonction.

```
int fCalculerSecondes(int, int , int) ;
void pAfficherHeure(int, int , int) ;
```

Seuls les types des paramètres sont indispensables pour le prototype : **il est cependant conseillé d'indiquer leurs noms pour faciliter la compréhension** :

```
int fCalculerSecondes(int pHeures, int pMin , int pSec);
void pAfficherHeure(int pHeures, int pMin , int pSec);
```

B. Valeurs par défaut des paramètres

Il est possible de **définir des valeurs par défaut** pour les paramètres **dans le prototype** d'une fonction.

```
int fCalculerSecondes(int pHeures, int pMin = 0, int pSec = 0);
```

Lors de l'appel :

```
nbre = fCalculerSecondes(12);
```

VI. Surcharge : notion de signature

Exemple : une fonction permettant d'effectuer la somme de nombres de types différents

```
// prototypes des fonctions
int fAdditionner(int pV1, int pV2) ;
double fAdditionner(double pV1, double pV2) ;

int main ()
{
// déclaration const. Et var. locales
int nbEnt1, nbEnt2;
double nbReel1, nbReel2;

   printf( "Entrez 2 nombres entiers séparés par un espace:");
   scanf("%d %d",&nbEnt1, &nbEnt2);
   printf( "le résultat est %d" , fAdditionner(nbEnt1, nbEnt2));

   printf( "Entrez 2 nombres réels séparés par un espace:");
   scanf("%f %f",&nbReel1, &nbReel2);
   printf( "le résultat est %d" , fAdditionner(nbReel1, nbReel2));

}
// Définition des fonctions
int fAdditionner(int pV1, int pV2)
{
   return (pV1 + pV2) ;
}
double fAdditionner(double pV1, double pV2)
{
   return (pV1 + pV2) ;
```

```
}
```

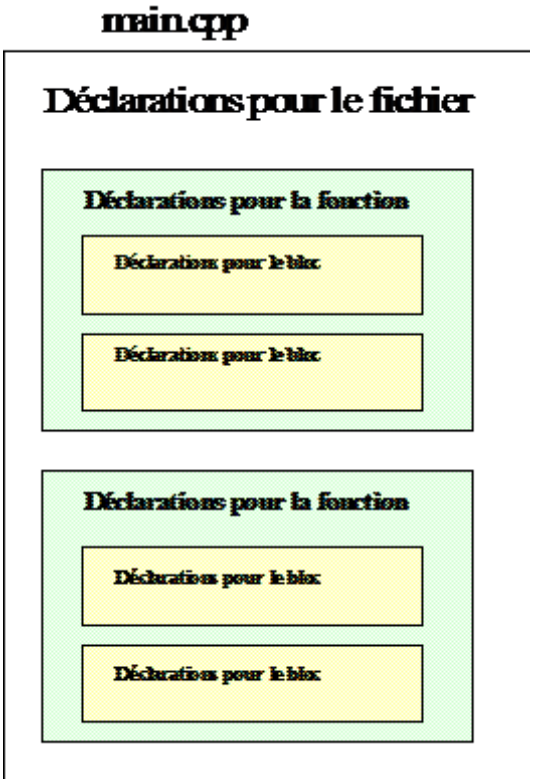
VII. Portée et visibilité des déclarations

La notion de **PORTEE** (anglais scope) fait référence à la région d'un programme à l'intérieur duquel une donnée (variable ou constante) est connue.

Anglais : scope (fr. étendue, portée)

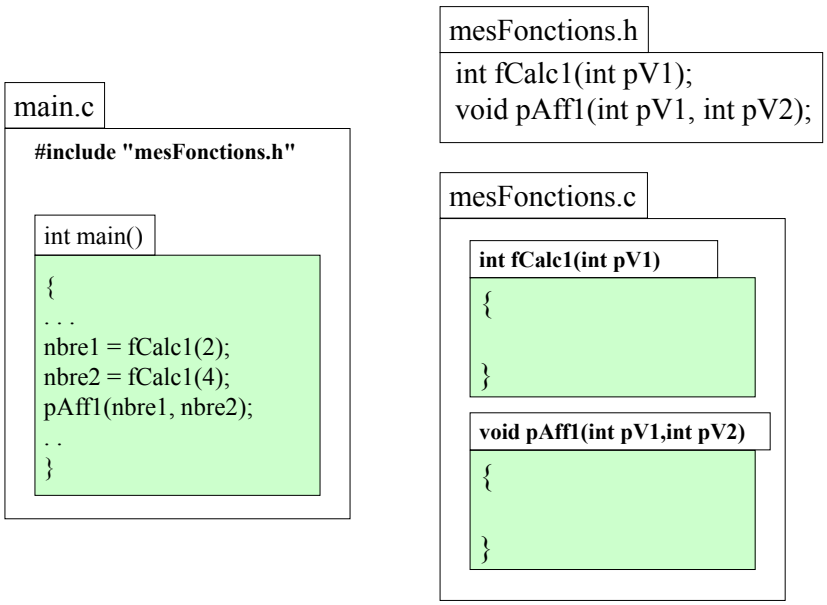
En C, on peut effectuer des déclarations :

- Au niveau **fichier**
  - **le fichier forme le bloc de niveau le plus élevé** : toute déclaration effectuée à ce niveau est connue dans toutes les fonctions du fichier et tous les blocs imbriqués (elle est globale au fichier) ;
  - Au trouve fréquemment à ce niveau les **prototypes** des fonctions utilisées dans ce fichier ;
- Au niveau **fonction**
  - Toute déclaration effectuée dans une **fonction** (locale à la fonction) est disponible à partir de sa déclaration, jusqu'à la fin de la fonction, et dans les blocs imbriqués ;
- Au niveau **bloc**
  - une déclaration peut être réalisée à n'importe quel endroit du code. Une déclaration réalisée dans un **bloc** est disponible dans ce bloc, à partir de sa déclaration jusqu'à la fin du bloc, **et dans les blocs imbriqués**.



VIII. Organisation des fichiers d'un programme C

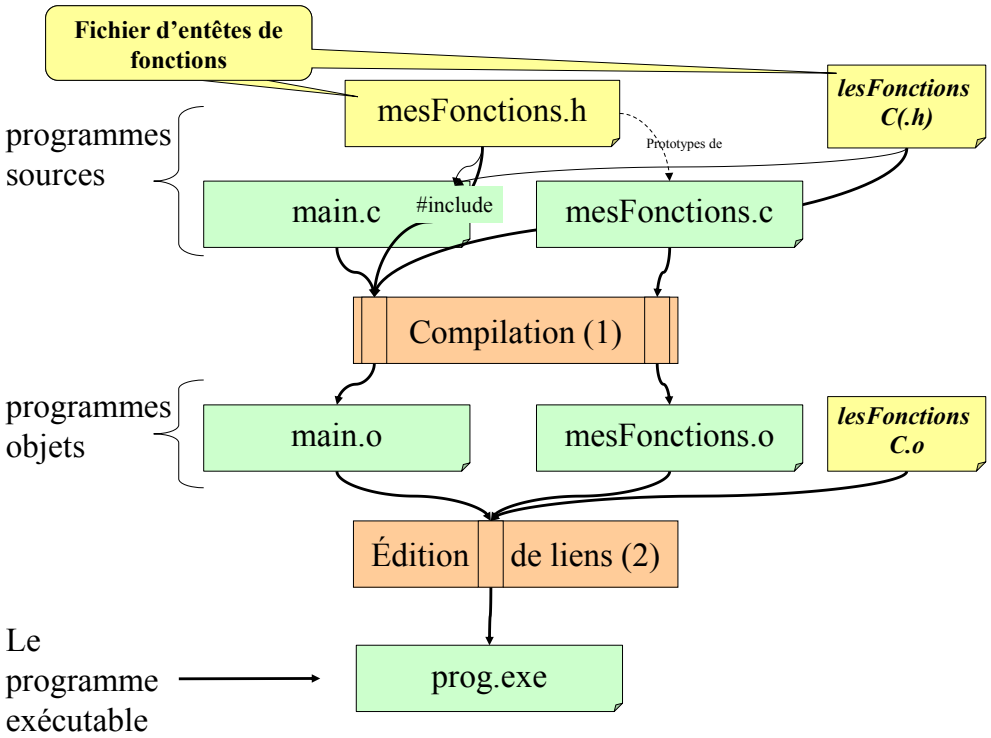
A. Packages et Organisation des fichiers sources



Exemple de fichier d'entête : (pour éviter les inclusions multiples, sources d'erreurs) :

```
#ifndef MESFONCTIONS_H
#define MESFONCTIONS _H
int fCalc1(int pV1) ; // prototype
void pAff1(int pV1, int pV2) ; // prototype
#endif
```

B. Compilation (compilation + édition de liens)



IX. Annexes

A. Tableaux et passage de paramètres

En C, les tableaux sont passés par référence ; en effet, le nom d'un tableau passé en argument est interprété comme l'adresse de son premier élément.

1. Passage d'un tableau et MODIFICATION DES VALEURS

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

const int MAX = 10; // ou DEFINE MAX 10;

void afficher (int tab[MAX])
{
    int i;
    for(i = 0; i<MAX; i++)
        printf("%d=%d / ",i, tab[i]);
}
void doubler (int tab[MAX])
{
    int i;
    for(i = 0; i<MAX; i++)
        tab[i] = tab[i]*2;
}
int main()
{
    int tab[MAX];
    int i;
    for(i = 0; i<MAX; i++)
        tab[i] = i;
    afficher(tab);
    doubler(tab);
    afficher(tab);
    return 0;
}
```

2. Passage d'un tableau et PROTECTION DES VALEURS

Le mot-clef **const** permet la **protection d'une variable** tableau **passée par référence** (= éviter que le sous-programme puisse le modifier).

```
void afficher (const int tab[MAX])
{
    int i;
    for(i = 0; i<MAX; i++)
        printf("%d=%d / ",i, tab[i]);
}
```

3. Passage de tableaux multidimensionnels

Dans le cas de tableaux multidimensionnels, il faut déclarer les dimensions du tableau passé en argument. **La première dimension peut être omise, mais les suivants doivent obligatoirement contenir le nombre d'élément** correspondant à chacune des dimensions.

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

const int MAX = 10; // ou DEFINE MAX 9;

void afficher(int tab[][MAX])
{
    int i, j;
    for(i = 0; i<MAX; i++)
    {
        printf("\n%d-",i);
        for(j = 0; j<MAX; j++)
            printf("%d ",tab[i][j]);
    }
 }

int main()
{
    int tab[MAX][MAX];
    int i, j;
    for(i = 0; i<MAX; i++)
        for(j = 0; j<MAX; j++)
            tab[i][j] = i;
    afficher(tab);
    return 0;
}
```