

CHAPITRE VIII

LES FONCTIONS

Objectifs

Général

⇒ Pouvoir développer un programme modulaire en langage C.

Spécifiques

- ❑ Comprendre la notion de modularisation de programmes;
 - ❑ Assimiler les notions de variables locales et variables globales.
 - ❑ Savoir manipuler et utiliser les fonctions dans un programme.
-

1. Introduction

La plupart des langages de programmation nous permettent de subdiviser nos programmes en sous-programmes, *fonctions* ou *procédures* plus simples et plus compacts. A l'aide de ces structures nous pouvons *modulariser* nos programmes pour obtenir des solutions plus efficaces et compréhensibles.

Exemple : Afficher un rectangle d'étoiles

Le programme suivant permet d'afficher à l'écran un rectangle de longueur L et de hauteur H, formé d'étoiles '*' :

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

int main(void)
{
    /* Prototypes des fonctions appelées par la fonction main */
    void RECTANGLE(int L, int H);
    /* Déclaration des variables locales de la fonction main */
    int L, H;
    /* Traitements */
    printf("Entrer la longueur (>= 1): ");
    scanf("%d", &L);
    printf("Entrer la hauteur (>= 1): ");
    scanf("%d", &H);
    /* Afficher un rectangle d'étoiles */
    RECTANGLE(L,H);
    getch();
    return 0;
}

void RECTANGLE(int L, int H)
{
    /* Prototypes des fonctions appelées */
    void LIGNE(int L);
    /* Déclaration des variables locales */
    int I;
    /* Traitements */
    /* Afficher H lignes avec L étoiles */
```

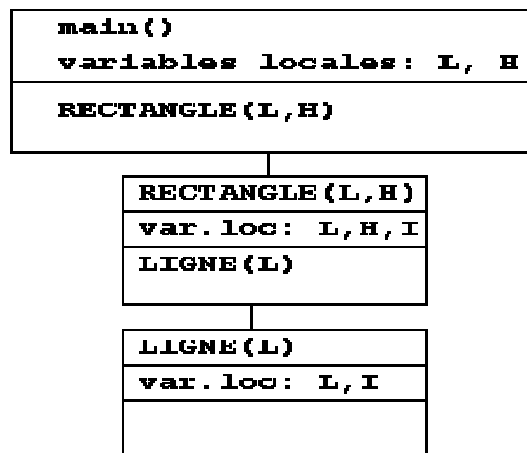
```

    for (I=0; I<H; I++)
        LIGNE(L);
}

void LIGNE(int L)
{
    /* Affiche à l'écran une ligne avec L étoiles */
    /* Déclaration des variables locales */
    int I;
    /* Traitements */
    for (I=0; I<L; I++)
        printf("*");
    printf("\n");
}

```

Schématiquement, nous pouvons représenter la hiérarchie des fonctions du programme comme suit:



2. La notion de portée des identificateurs

2.1. Variables locales

Les variables déclarées dans un bloc d'instructions sont *uniquement visibles à l'intérieur de ce bloc*. On dit que ce sont des **variables locales** à ce bloc.

Exemple 1

```

void AFFICH_NOM(void);
{
    char NOM[20];
    printf("Introduisez votre nom : ");
    gets(NOM);
    printf("Votre nom est %s !\n", NOM);
}

```

Exemple 2

```

if (N>0)
{
    int I;
    for (I=0; I<N; I++)
        ...
}

```

2.2. Variables globales

Les variables déclarées au début du fichier, à l'extérieur de toutes les fonctions *sont disponibles à toutes les fonctions du programme*. Ce sont alors des *variables globales*. En général, les variables globales sont déclarées immédiatement derrière les instructions `#include` au début du programme.

Exemple :

```
#include <stdio.h>
#include <conio.h>
int STATUS;
```

3. Modularisation de programmes

3.1. Exemple d'un programme modulaire

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
    /* Prototypes des fonctions appelées */
    int ENTREE(void);
    int MAX(int N1, int N2);
    /* Déclaration des variables */
    int A, B;
    /* Traitement avec appel des fonctions */
    A = ENTREE();
    B = ENTREE();
    printf("Le maximum est %d\n", MAX(A,B));
    getch();
    return 0;
}

/* Définition de la fonction ENTREE */
int ENTREE(void)
{
    int NOMBRE;
    printf("Entrez un nombre entier : ");
    scanf("%d", &NOMBRE);
    return NOMBRE;
}

/* Définition de la fonction MAX */
int MAX(int N1, int N2)
{
    if (N1>N2)
        return N1;
    else
        return N2;
}
```

3.2. Définition d'une fonction

Exemple 1 : La fonction MAX est du type `int` et elle a besoin de deux paramètres du type `int`. Le résultat de la fonction MAX peut être intégré dans d'autres expressions.

```
int MAX(int N1, int N2)
{
```

```
    if (N1>N2)
        return N1;
    else
        return N2;
}
```

Exemple 2 : La fonction PI fournit un résultat rationnel du type **float**. La liste des paramètres de PI est déclarée comme **void** (vide), c.-à-d. PI n'a pas besoin de paramètres et il faut l'appeler par : PI()

```
float PI(void)
{
    return 3.1415927;
}
```

3.3. Déclaration d'une fonction

De façon analogue aux déclarations de variables, nous pouvons déclarer une fonction localement ou globalement. La définition des fonctions joue un rôle spécial pour la déclaration. En résumé, nous allons considérer les règles suivantes :

- **Déclaration locale :**

Une fonction peut être déclarée localement *dans la fonction qui l'appelle* (avant la déclaration des variables). Elle est alors disponible à cette fonction.

- **Déclaration globale :**

Une fonction peut être déclarée globalement *au début du programme* (derrière les instructions **#include**). Elle est alors disponible à toutes les fonctions du programme.

4. Renvoyer un résultat

Par définition, toutes les fonctions fournissent un résultat d'un type que nous devons déclarer. Une fonction peut renvoyer une valeur d'un type simple ou l'adresse d'une variable ou d'un tableau.

```
    return <expression>;
```

5. Paramètres d'une fonction

Les paramètres d'une fonction sont simplement des **variables locales** qui sont initialisées par les **valeurs** obtenues lors de l'appel.

5.1. Passage des paramètres par valeur

Exemple : Nous voulons écrire une fonction PERMUTER qui échange le contenu de deux variables du type **int**. En première approche, nous écrivons la fonction suivante :

```
void PERMUTER (int A, int B)
{
    int AIDE;
    AIDE = A;
    A = B;
    B = AIDE;
}
```

5.2. Passage de l'adresse d'une variable

Pour pouvoir modifier le contenu de X et de Y, la fonction PERMUTER a besoin des adresses de X et Y. En utilisant des pointeurs, nous écrivons une deuxième fonction :

```
void PERMUTER (int *A, int *B)
{
    int AIDE;
    AIDE = *A;
    *A = *B;
    *B = AIDE;
}
```

6. Exercices d'application

Exercice 8.1

Énoncé

Soient les fonctions **main**, **PI** et **SURFACE** définies par :

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
    double R;
    printf("Introduire le rayon du cercle : ");
    scanf("%lf", &R);
    printf("La surface du cercle est %f. \n", SURFACE(R));
    return 0;
}

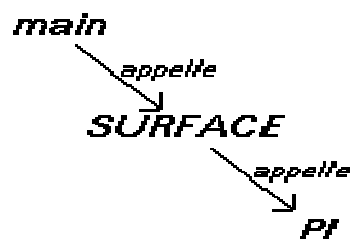
double PI(void)
{
    return 3.14159265;
}

double SURFACE(double RAYON)
{
    return PI()*RAYON*RAYON;
}
```

- a) Etablir la hiérarchie des appels pour les trois fonctions.
- b) Définir les fonctions et ajouter les déclarations manquantes.

Solution

a)



b) Plusieurs solutions sont envisageable :

- Déclarations locales des fonctions et définition 'top-down'.
- Définition 'bottom-up' sans déclarations.
- Déclaration globale des fonctions et définition 'top-down'.

Exercice 8.2

Énoncé

- a) Ecrire une fonction MIN et une fonction MAX qui déterminent le minimum et le maximum de deux nombres réels.
- b) Ecrire un programme se servant des fonctions MIN et MAX pour déterminer le minimum et le maximum de quatre nombres réels entrés au clavier.

Solution

```
int main(void)
{
    /* Prototypes des fonctions appelées */
    double MIN(double X, double Y);
    double MAX(double X, double Y);
    /* Variables locales */
    double A,B,C,D;
    /* Traitements */
    printf("Introduire 4 réels : ");
    scanf("%lf %lf %lf %lf", &A, &B, &C, &D);
    printf("Le minimum des 4 réels est %f \n", MIN( MIN(A,B), MIN(C,D)) );
    printf("Le maximum des 4 réels est %f \n", MAX( MAX(A,B), MAX(C,D)) );
    return 0;
}

double MIN(double X, double Y)
{
    if (X<Y)
        return X;
    else
        return Y;
}

double MAX(double X, double Y)
{
    if (X>Y)
        return X;
    else
        return Y;
}

/* ou bien */ /*
double MIN(double X, double Y)
{
    return (X<Y) ? X : Y;
}

double MAX(double X, double Y)
{
    return (X>Y) ? X : Y;
} */
```
