8.3. Déclaration et définition de fonctions

En général, le nom d'une fonction apparaît à trois endroits dans un programme:

1) lors de la **déclaration**

2) lors de l'**appel**

3) lors de la **définition**

8.3.1. Définition d'une fonction

Dans la définition d'une fonction, nous indiquons:

- 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
- les instructions à exécuter

Définition d'une fonction en langage algorithmique

```
fonction <NomFonct> (<NomPar1>, <NomPar2>, ...):<TypeRés>

| <déclarations des paramètres>
| <déclarations locales>
| <instructions>
ffonction
```

Définition d'une fonction en C

```
<TypeRés> <NomFonct> (<TypePar1> <NomPar1>,
                      <TypePar2> <NomPar2>, ... )
{
 <déclarations locales>
 <instructions>
}
```

Remarquez qu'il n'y a pas de point-virgule derrière la définition des paramètres de la fonction.

8.3.2. Déclaration d'une fonction

En C, il faut déclarer chaque fonction avant de pouvoir l'utiliser.
Prototype d'une fonction

La déclaration d'une fonction se fait par un ***prototype*** de la fonction qui indique uniquement le type des données transmises et reçues par la fonction.

Déclaration : Prototype d'une fonction

<TypeRés> <NomFonct> (<TypePar1>, <TypePar2>, ...);
ou bien
<TypeRés> <NomFonct> (<TypePar1> <NomPar1>,
<TypePar2> <NomPar2>, ...);

Règles pour la déclaration des fonctions

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:

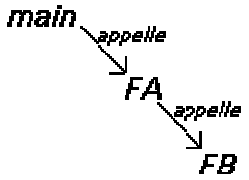
Une fonction peut être déclarée localement *dans la fonction qui l'appelle* (avant la déclaration des variables). Elle est alors disponible à cette fonction.

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.

8.3.3. Discussion d'un exemple

Considérons la situation suivante:



a) Déclarations locales des fonctions et définition 'top-down'

La définition 'top-down' suit la hiérarchie des fonctions:

Nous commençons par la définition de la fonction principale **main**, suivie des sous-programmes FA et FB. Nous devons déclarer explicitement FA et FB car leurs définitions suivent leurs appels.

```
/* Définition de main */
main()
{
  /* Déclaration locale de FA */
  int FA (int X, int Y);
  ...
  /* Appel de FA */
  I = FA(2, 3);
  ...
}

/* Définition de FA */
int FA(int X, int Y)
{
  /* Déclaration locale de FB */
  int FB (int N, int M);
  ...
  /* Appel de FB */
  J = FB(20, 30);
  ...
}

/* Définition de FB */
int FB(int N, int M)
{
}
```

8.4. 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.

Pour fournir un résultat en quittant une fonction, nous disposons de la commande **return**:

La commande return L'instruction

return <expression>;

a les effets suivants:

- évaluation de l'<expression>
- conversion automatique du résultat de l'expression dans le type de la fonction
- renvoi du résultat
- terminaison de la fonction

Exemples

La fonction CARRE du type **double** calcule et fournit comme résultat le carré d'un réel fourni comme paramètre.

```
double CARRE(double X)
{
 return X*X;
}
```

8.5. Paramètres d'une fonction

8.5.1. Généralités

Conversion automatique

Lors d'un appel, le nombre et l'ordre des paramètres doivent nécessairement correspondre aux indications de la déclaration de la fonction. Les paramètres sont automatiquement convertis dans les types de la déclaration avant d'être passés à la fonction.

Exemple

Le prototype de la fonction **pow** (bibliothèque *<math>*) est déclaré comme suit:

double pow (double, double);
Au cours des instructions,

```
int A, B;
...
A = pow (B, 2);
```

8.5.2. Passage des paramètres par valeur

En C, le passage des paramètres se fait toujours par la valeur, c.-à-d. les fonctions n'obtiennent que les ***valeurs*** de leurs paramètres et n'ont pas d'accès aux variables elles-mêmes.

Les paramètres d'une fonction sont à considérer comme des ***variables locales*** qui sont initialisées automatiquement par les valeurs indiquées lors d'un appel.

8.5.3. Passage de l'adresse d'une variable

Comme nous l'avons constaté ci-dessus, une fonction n'obtient que les valeurs de ses paramètres.

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***.

On peut alors atteindre la variable à l'aide du pointeur.

Discussion d'un exemple

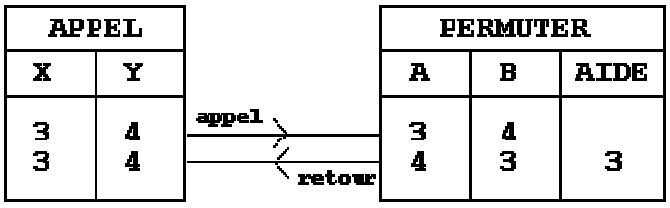
```
void PERMUTER (int A, int B)
{
 int AIDE;
 AIDE = A;
 A = B;
 B = AIDE;
}
```

Nous appelons la fonction pour deux variables X et Y par:

PERMUTER(X, Y);

Résultat: X et Y restent inchangés !

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



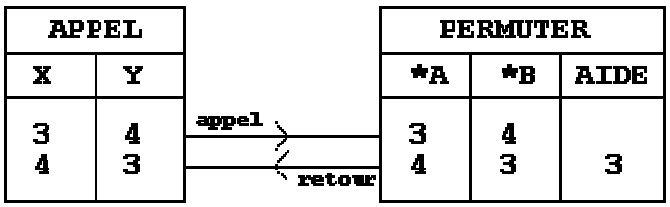
Pour pouvoir modifier le contenu de X et de Y, la fonction PERMUTER a besoin des adresses de X et Y. En utilisant des pointeurs, nous écrivons une deuxième fonction:

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

Nous appelons la fonction par:

PERMUTER(&X, &Y);

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



Exercice 1

Ecrire un programme se servant d'une fonction MOYENNE du type **float** pour afficher la moyenne arithmétique de deux nombres réels entrés au clavier.

```
#include <stdio.h>

main()
{
 /* Prototypes des fonctions appelées */
 float MOYENNE(float X, float Y);
 /* Variables locales */
 float A,B;
 /* Traitements */
 printf("Introduire deux nombres : ");
 scanf("%f %f", &A, &B);
 printf("La moyenne arithmétique de %.2f et %.2f est %.4f\n",
                                  A, B, MOYENNE(A,B));
 return 0;
}

float MOYENNE(float X, float Y)
{
 return (X+Y)/2;
}
```

Exercice 2

En mathématiques, on définit la fonction factorielle de la manière suivante:

0! = 1

n! = n*(n-1)*(n-2)* ... * 1 (pour n>0)

Ecrire une fonction FACT du type **double** qui reçoit la valeur N (type **int**) comme paramètre et qui fournit la factorielle de N comme résultat. Ecrire un petit programme qui teste la fonction FACT.

```
#include <stdio.h>

main()
{
 /* Prototypes des fonctions appelées */
 double FACT(int N);
 /* Variables locales */
 int N;
 /* Traitements */
 printf("Introduire un nombre entier N : ");
 scanf("%d", &N);
 printf("La factorielle de %d est %.0f \n",N , FACT(N));
 return 0;
}

double FACT(int N)
{
  /* Comme N est transmis par valeur, N peut être */
 /* modifié à l'intérieur de la fonction. */
 double RES;
 for (RES=1.0 ; N>0 ; N--)
     RES *= N;
 return RES;
}
```