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

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

Exemple

```
int MAX(int N1, int N2)
{
 if (N1>N2)
    return N1;
 else
    return N2;
}
```

Remarques

- Une fonction peut fournir comme résultat:
  - un type arithmétique,
  - une structure (définie par **struct**)
  - un pointeur,
  - **void** (la fonction correspond alors à une 'procédure').

Une fonction **ne peut pas** fournir comme résultat des tableaux, des chaînes de caractères ou des fonctions.

- Le type par défaut est **int**; autrement dit: si le type d'une fonction n'est pas déclaré explicitement, elle est automatiquement du type **int**.
- Il est interdit de définir des fonctions à l'intérieur d'une autre fonction.
- Chaque fonction doit être déclarée ou définie avant d'être appelée.

2. Déclaration d'une fonction

Déclaration : Prototype d'une fonction

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

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

- **Déclaration locale:** Une fonction peut être déclarée localement *dans la fonction qui l'appelle* (avant la déclaration des variables).

- **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.

- **Déclaration implicite par la définition:**
La fonction est automatiquement disponible à toutes les fonctions qui suivent sa définition.

3. Les valeurs de retour

La commande return

L'instruction **return <expression>;** évalue l'<expression>, convertit automatique le résultat de l'expression dans le type de la fonction, renvoie le résultat et termine la fonction

Exemples

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

double TAN(double X)
{
 if (cos(X) != 0)
    return sin(X)/cos(X);
 else
    printf("Erreur !\n");
}
```

```
        double X, COT;

        printf("Le carre de %f est %f \n", X, CARRE(X));
        printf("La tangente de %f est %f \n", X, TAN(X));
        COT = 1/TAN(X);
```

Le type void

La procédure LIGNE affiche L étoiles dans une ligne:

```
void LIGNE(int L)
{
 /* Déclarations des variables locales */
 int I;
 /* Traitements */
 for (I=0; I<L; I++)
```

```
    printf("*");
 printf("\n");
}
```

4. Notion de bloc et la portée des dentificateurs

Un bloc d'instruction est encadré d'accolade « { }» est composé de 2 parties :

- Déclarations des variables locales
- Instructions

a- 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.
Une variable déclarée à l'intérieur d'un bloc cache toutes les variables du même nom des blocs qui l'entourent.

b- 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 directives ***#include*** au début du programme.

Les variables déclarées au début de la fonction principale *main* ne sont pas des variables globales, mais elles sont locales à *main*.

```
#include <stdio.h>
int i ;  (vatiable globale)
main()
{ void affiche(void) ;
for (i=1 ; i<=5 ; i++)
affiche() ;
}
void affiche(void)
{ printf ("bonjour %d fois\n", i) ;}
```

```
Bonjour 1 fois
Bonjour 2 fois
Bonjour 3 fois
Bonjour 4 fois
Bonjour 5 fois
```

5. Passage des paramètres par valeur

En C, le passage des paramètres se fait toujours par la valeur.

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

6. Passage de l'adresse d'une variable

Discussion d'un exemple

Soit la fonction Permuter définie ci-dessous :

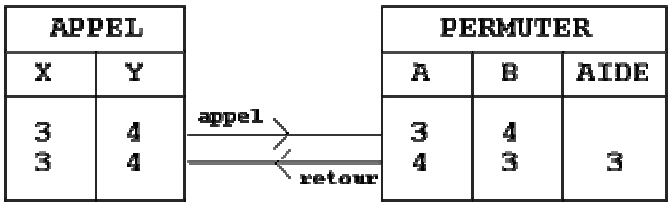
```
void PERMUTER (int A, int B)
{
 int AIDE;
 AIDE = A;
 A = B;
 B = AIDE;
}
```

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:

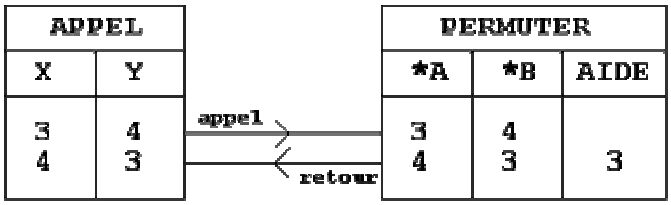
```
void PERMUTER (int *A, int *B)
{
 int AIDE;
 AIDE = *A;
 *A = *B;
 *B = AIDE;
}
```

Nous appelons la fonction par:

```
PERMUTER(&X, &Y);
```

Résultat: Le contenu des variables X et Y est échangé !

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.



7. Passage de l'adresse d'un tableau à une dimension

Dans la liste des paramètres d'une fonction, on peut déclarer un tableau par le nom suivi de crochets,

<type> <nom>[]

ou simplement par un pointeur sur le type des éléments du tableau:

<type> *<nom>

Exemple

La fonction **strlen** calcule et retourne la longueur d'une chaîne de caractères fournie comme paramètre:

```
int strlen(char *S)
{
 int N;
 for (N=0; *S != '\0'; S++)
        N++;
 return N;
}
```

A la place de la déclaration de la chaîne comme

char *S

on aurait aussi pu indiquer

char S[]

Appel

Lors d'un appel, l'adresse d'un tableau peut être donnée par le nom du tableau, par un pointeur ou par l'adresse d'un élément quelconque du tableau.

Exemple

Après les instructions,

```
char CH[] = "Bonjour !";
char *P;
P = CH;
```

nous pouvons appeler la fonction **strlen** définie ci-dessus par:

```
strlen(CH)      /* résultat: 9 */
strlen(P)       /* résultat: 9 */
strlen(&CH[4])  /* résultat: 5 */
strlen(P+2)     /* résultat: 7 */
strlen(CH+2)    /* résultat: 7 */
```

Exemple

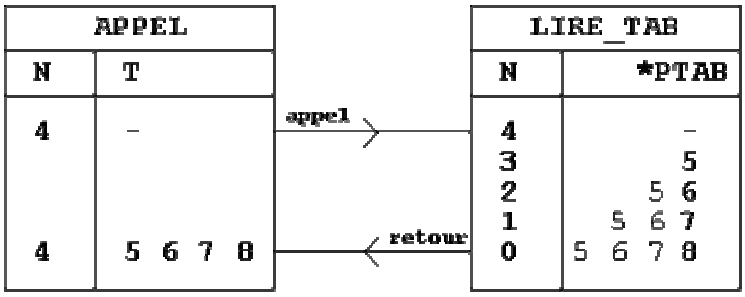
La fonction LIRETAB lit N données pour un tableau (unidimensionnel) du type **int** et les mémorise à partir de l'adresse indiquée par le pointeur PTAB. PTAB et N sont fournis comme paramètres.

```
void LIRE_TAB(int N, int *PTAB)
{
 printf("Entrez %d valeurs : \n", N);
 while(N)
   {
    scanf("%d", PTAB++);
    N--
   }
}
```

Dans l'appel de la fonction nous utilisons en général le nom du tableau:

```
LIRE_TAB(4, T);
```

Nous obtenons alors les grilles suivantes:



Exercices

1. Ecrire deux fonctions qui calculent la valeur X^N pour une valeur réelle X (type **double**) et une valeur entière positive N (type **int**) :

a) EXP1 retourne la valeur X^N comme résultat.

b) EXP2 affecte la valeur X^N à X.

Ecrire un programme qui teste les deux fonctions à l'aide de valeurs lues au clavier.

2. Ecrire la fonction NCHIFFRES du type **int** qui obtient une valeur entière N (positive ou négative) du type **long** comme paramètre et qui fournit le nombre de chiffres de N comme résultat.

Ecrire un petit programme qui teste la fonction NCHIFFRES:

Exemple:

Introduire un nombre entier : 6457392
Le nombre 6457392 a 7 chiffres.

3. Ecrire la fonction SOMME_TAB qui calcule la somme des N éléments d'un tableau TAB du type **int**. N et TAB sont fournis comme paramètres; la somme est retournée comme résultat du type **long**.

4. Ecrire la fonction NMOTS_CH qui retourne comme résultat le nombre de mots contenus dans une chaîne de caractères CH. Utiliser une variable logique, la fonction **isspace** et une variable d'aide N.

5. Ecrire la fonction CH_ENTIER qui retourne la valeur numérique d'une chaîne de caractères représentant un entier (positif ou négatif) du type **long**. Si la chaîne ne représente pas une valeur entière correcte, la fonction arrête la conversion et fournit la valeur qu'elle a su reconnaître jusqu'à ce point.