Chapitre 4

Les fonctions

I. Définition d'une fonction

En langace C, une fonction est définie comme suit :

```
TypeRésultatRetourné   NomFonction (TypePar1 NomPar1,
   TypePar2 NomPar2,... )
{
   déclarations locales;
   instructions;
   return(Expr);
}
```

Exemple

```
int somme_carre( int i , int j )
{
int resultat;
resultat=i*i+j*j;
return(resultat);
}
```

Remarques

- Le nom de la fonction doit respecter les mêmes règles que les noms de variables.
- Le type de la valeur retourné par la fonction par défaut est int ; autrement dit : si le type d'une fonction n'est pas déclaré explicitement, elle est automatiquement du type int.
- Si une fonction ne fournit pas de résultat, il faut indiquer void (vide) comme type du résultat.
- Si une fonction fournit un résultat, il est nécessaire que sa définition contient l'instruction return(expression).
- Une fonction ne peut pas fournir comme résultat des tableaux, des chaînes de caractères ou des fonctions.
- Si une fonction n'a pas de paramètres, on peut déclarer la liste des paramètres comme (void) ou simplement comme ().
- Il est interdit de définir des fonctions à l'intérieur d'une autre fonction.
- l'ordre des définitions des fonctions dans le texte du programme ne joue pas de rôle, mais chaque fonction doit être déclarée ou définie avant d'être appelée.

2. Appel d'une fonction

L'appel d'une fonction se fait par son nom suivi de la liste des paramètres effectifs mis entre parenthèse et séparés par des virgules. Ces paramètres correspondre en nombre et en type aux paramètres formels indiqués dans le prototype de la fonction.

```
nom_fonction(param_effectif1, param_effectif2, ...);
```

3. Déclaration d'une fonction

Avant d'utiliser une fonction, on doit la déclarer. La déclaration se fait selon la syntaxe suivante:

TypeRésultatRetourné NomFonction (TypePar1 NomPar1, TypeParN NomParN,...);

Une fonction peut être déclarée soit :

- Localement dans la fonction appelante, elle sera disponible uniquement dans cette fonction.
- Globalement au début du programme, elle sera disponible à toutes les fonctions du programme.

Remarque : une fonction peut être utilisée sans déclaration à condition qu'elle soit définie avant son appel.

4. Passage des paramètres par valeur

Exemple : une fonction qui calcule le factoriel d'un nombre
N>0

```
int factoriel (int N)
{
   int fact= l ;
   while (N>0)
  {
      fact=fact*N;
      N--;
  }
  return(fact);
}
```

- fonction qui calcule et affiche le factoriel des 10 premiers entiers

```
void calcul_fact(void)
{
   int L;
   for (L=1; L<10; L++)
   printf("Fact(%d)=%d\n",L,factoriel(L));
}
```

- Au moment de l'appel, la valeur de L est copiée dans N.
- La variable N peut donc être décrémentée à l'intérieur de factoriel, sans influencer la valeur originale de L.
- Le passage des paramètres par valeur ne permet pas à une fonction de changer le contenu des paramètres effectifs.

5. Passage de l'adresse d'une variable (passage par 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, il faut procéder comme suit :

- La fonction appelante doit fournir l'adresse de la variable et
- La fonction appelée doit déclarer le paramètre comme pointeur.

Exemple : Fonction permettant d'échanger la valeur de 2 variables :

```
#include <stdio.h>
void PERMUTER (int *x,int *y)
{
  int aux;
  aux=*x;
  *x=*y;
  *y=aux;
}
void main()
{
  int A=5,B=8;
  PERMUTER(&A,&B);
  printf("A=%d\n",A);
  printf("B=%d\n",B);
}
```

Lors de l'appel, les adresses de A et B sont copiées dans les pointeurs x et y. PERMUTER échange ensuite les contenus des adresses indiquées par les pointeurs x et y.

Exercice

Un nombre parfait est un entier positif supérieur à 1, égal à la somme de ses diviseurs ; on compte 1 comme diviseur, mais on ne compte pas comme diviseur le nombre lui-même.

Exemple : 6 est un nombre parfait puisque : 6=3+2+1.

1. Ecrire une fonction qui prend en argument un nombre entier n et retourne si n est parfait ou non.
2. Ecrire une fonction qui prend en argument un entier positif n et qui affiche tous les nombres parfaits inférieurs à ce nombre