CESI_
ÉCOLE D'INGÉNIEURS

A3 FISA S3E – Langage C

TP 3 – Les fonctions

1. DEFINITION

Comme dans la plupart des langages, on peut en C **découper un programme en plusieurs fonctions.** Une seule de ces fonctions existe obligatoirement ; c'est la fonction principale appelée ***main***. Cette fonction principale peut, éventuellement, appeler une ou plusieurs fonctions secondaires. De même, chaque fonction secondaire peut appeler d'autres fonctions secondaires ou s'appeler elle-même (dans ce dernier cas, on dit que la fonction est récursive).

La définition d'une fonction est la donnée du texte de son algorithme, qu'on appelle corps de la fonction. Elle est de la forme :

```
type nom-fonction (type-1 arg-1,...,type-n arg-n)
{
[déclarations de variables locales ]
 liste d'instructions
 return c ;
}
```

La première ligne de cette définition est **l'*en-tête*** de la fonction. Dans cet en-tête, ***type*** désigne le type de la fonction, c'est-à-dire le type de la valeur qu'elle retourne. Contrairement à d'autres langages, il n'y a pas en C de notion de procédure ou de sous-programme. Une fonction qui ne renvoie pas de valeur est une fonction dont le type est spécifié par le mot-clef **void**.

Les arguments de la fonction sont appelés ***paramètres formels***, par opposition aux *paramètres effectifs* qui sont les paramètres avec lesquels la fonction est effectivement appelée. Les paramètres formels peuvent être de n'importe quel type. Leurs identificateurs n'ont d'importance qu'à l'intérieur de la fonction. Enfin, si la fonction ne possède pas de paramètres, on remplace la liste de paramètres formels par le mot-clef ***void***.

Le corps de la fonction débute éventuellement par des déclarations de variables, qui sont locales à cette fonction. Il se termine par l'*instruction de retour à la fonction appelante*, return, dont la syntaxe est : return(***expression***);

La valeur de ***expression*** est la valeur que retourne la fonction. Son type doit être le même que celui qui a été spécifié dans l'en-tête de la fonction. Si la fonction ne retourne pas de valeur (fonction de type void), sa définition s'achève par : ***return***;

Plusieurs instructions return peuvent apparaître dans une fonction. Le retour au programme appelant sera alors provoqué par le premier return rencontré lors de l'exécution. Voici quelques exemples de définitions de fonctions :

```
int produit (int a, int b)
{
Int c ;
C=a*b ;
  return(c);
}

int puissance (int a, int n)
{
  if (n == 0)
    return(1);
  return(a * puissance(a, n-1));
}


void imprime_tab (int *tab, int nb_elements)
{
  int i;
  for (i = 0; i < nb_elements; i++)
    printf("%d \t",tab[i]);
  printf("\n");
  return 0;
}
```

2. APPEL D'UNE FONCTION

L'appel d'une fonction se fait par l'expression :

nom-fonction(para-1,para-2,...,para-n)

L'ordre et le type des paramètres effectifs de la fonction doivent concorder avec ceux donnés dans l'en-tête de la fonction. Les paramètres effectifs peuvent être des expressions. La **virgule** qui sépare deux paramètres effectifs est un simple signe de ponctuation ; il ne s'agit pas de l'opérateur *virgule*. Cela implique en particulier que l'ordre d'évaluation des paramètres effectifs n'est pas assuré et dépend du compilateur.

3. DECLARATION D'UNE FONCTION

Le C n'autorise pas les fonctions imbriquées. La définition d'une fonction secondaire doit donc être placée soit avant, soit après la fonction principale *main*. Toutefois, il est indispensable que le compilateur ``connaisse'' la fonction au moment où celle-ci est appelée. Si une fonction est définie après son premier appel (en particulier si sa définition est placée après la fonction *main*), elle doit impérativement être déclarée au préalable. Une fonction secondaire est déclarée par son *prototype*, qui donne le type de la fonction et celui de ses paramètres, sous la forme :

```
type nom-fonction(type-1,...,type-n);
```

Les fonctions secondaires peuvent être déclarées indifféremment avant ou au début de la fonction *main*. Par exemple, on écrira

```
int puissance (int, int );

int puissance (int a, int n)
{
  if (n == 0)
    return(1);
 return(a * puissance(a, n-1));
}


main()

{
  int a = 2, b = 5;
  printf("%d\n", puissance(a,b));
}
```

Même si la déclaration est parfois facultative (par exemple quand les fonctions sont définies avant la fonction *main* et dans le bon ordre), elle seule permet au compilateur de vérifier que le nombre et le type des paramètres utilisés dans la définition concordent bien avec le *prototype*. De plus, la présence d'une déclaration permet au compilateur de mettre en place d'éventuelles conversions des paramètres effectifs, lorsque la fonction est appelée avec des paramètres dont les types ne correspondent pas aux types indiqués dans le prototype. Ainsi les fichiers d'extension .h de la librairie standard (fichiers headers) contiennent notamment les prototypes des fonctions de la librairie standard. Par exemple, on trouve dans le fichier *math.h* le prototype de la fonction pow (élévation à la puissance) :

```
extern  double pow(double , double );
```

La directive au préprocesseur : `#include <math.h>`

permet au préprocesseur d'inclure la déclaration de la fonction ***pow*** dans le fichier source. Ainsi, si cette fonction est appelée avec des paramètres de type **int**, ces paramètres seront convertis en double lors de la compilation.

Par contre, en l'absence de directive au préprocesseur, le compilateur ne peut effectuer la conversion de type. Dans ce cas, l'appel à la fonction **pow** avec des paramètres de type **int** peut produire un résultat faux !

4. DUREE DE VIE DES VARIABLES

Les variables manipulées dans un programme C ne sont pas toutes traitées de la même manière. En particulier, elles n'ont pas toutes la même *durée de vie*. On distingue deux catégories de variables.

Les variables permanentes (ou statiques) : Une variable permanente occupe un emplacement en mémoire qui reste le même durant toute l'exécution du programme. Cet emplacement est alloué une fois pour toutes lors de la compilation, appelée *segment de données*. Par défaut, les variables permanentes sont initialisées à zéro par le compilateur. Elles sont caractérisées par le mot-clef static.

Les variables temporaires : Les variables temporaires se voient allouer un emplacement en mémoire de façon dynamique lors de l'exécution du programme. Elles ne sont pas initialisées par défaut. Leur emplacement en mémoire est libéré par exemple à la fin de l'exécution d'une fonction secondaire, appelée *segment de pile*.

4.1 Variables globales

On appelle *variable globale* une variable déclarée en dehors de toute fonction. Une variable globale est connue du compilateur dans toute la portion de code qui suit sa déclaration. Les variables globales sont systématiquement permanentes.

4.2 Variables locales

On appelle *variable locale* une variable déclarée à l'intérieur d'une fonction (ou d'un bloc d'instructions) du programme. Par défaut, les variables locales sont temporaires. Quand une fonction est appelée, elle place ses variables locales dans la pile. A la sortie de la fonction, les variables locales sont dépilées et donc perdues.

5. LES FONCTIONS RECURSIVES

Le Langage C autorise la récursivité des appels de fonctions. Cela veut dire qu'une fonction comporte, dans sa définition, au moins un appel à elle-même.

Voici l'exemple le plus classique d'une fonction calculant une factorielle de manière récursive.

```
int factorielle(int n)
{
        if (n ==1) return1;
        return n*factorielle(n-1);
}
```

Exercices :

1- Écrire une fonction distance ayant comme paramètres 4 doubles x_a, y_a,$x_b$et $y_b$qui représentent les coordonnées de deux points A et B et qui renvoie la distance AB. Tester cette fonction.

2- Écrire une fonction qui renvoie 1 si un nombre entier passé en paramètre est impair, 0 sinon. Son prototype est donc : **int estImpair(int nb);**

Écrire également son programme de test (main).

3- Ecrire une fonction ***void tableMultiplication()*** qui demande "Quelle table de multiplication voulez-vous, tapez 0 (zéro) pour sortir ?". Si le caractère saisi est compris entre '1' et '9' alors on fera afficher la table correspondante puis on réitérera le processus, sinon on affichera "ce n'est pas dans les possibilités du programme, recommencez !" et on réitérera le processus.

4- **Echange de deux valeurs :** Ecrire une fonction Echange qui réalise l'échange de deux valeurs entières passées en argument. Quel problème constatez-vous? Comment peut-on résoudre ce problème?

5- **Les fonctions récursives :**

a. Ecrire la fonction qui calcule la somme des n premiers carrés. Exemple : si n = 3 ; la fonction calculera $1^2 + 2^2 + 3^2$.

b. Ecrire la fonction qui permet de calculer, la multiplication de deux entiers par additions successives.

c. Ecrire la fonction qui permet de dire si un entier est pair ou impair, en supposant que les seules opérations possibles sont : la comparaison avec 0 et la comparaison avec 1.

6- Découverte

Dans cette partie seul le mécanisme de déclaration et d'utilisation des fonctions est important.

a. Ecrire une fonction chargée de calculer le carré d'un nombre réel (sans saisie ni affichage) ?
b. Utiliser cette fonction dans un programme test qui saisit un nombre, appelle la fonction pour l'élever au carré et affiche le résultat.
c. Modifier la fonction précédente et son programme test pour élever « x » à la puissance entière « y » ?

7- Les tableaux en C

a. Dans un programme principal, déclarer un tableau de 10 entiers.
b. Initialiser le tableau avec des valeurs aléatoires comprises entre 0 et 20.
c. Implémenter et tester à chaque fois les fonctions suivantes :
    - Fonction ***afficher*** qui permet d'afficher les éléments du tableau.
    - Fonction ***calculerMoyenne*** qui permet de calculer la moyenne des éléments du tableau.
    - Fonction ***trouverMin*** qui permet de trouver la valeur minimum du tableau.
    - Fonction ***inverserTableau*** qui inverse les éléments du tableau (penser _a une variante qui n'utilise pas de tableau supplémentaire).