Séance 9 : Les fonctions

Les fonctions en langage C permettent de regrouper certains traitements. Une fonction prend en entrée un certain nombre de variables et retourne une valeur. Nous avons déjà vu au travers des fonctions « `printf` » et « `scanf` », nous allons voir maintenant comment déclarer et utiliser nos propres fonctions.

Exemple d'utilisation

Voici un premier exemple d'utilisation d'une fonction :

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


int addition( int a, int b)


{
     int resultat;

     resultat  = a + b;

     return resultat;
}

int main(int argc, char **argv)


{
     int operation;

     operation = addition( 5, 14);

     printf( "Le resultat est %d\n", operation);


     return(EXIT_SUCCESS);
}
```

Dans cet exemple, nous définissons la fonction « `addition` ». Cette fonction prend en entrée deux entiers et retourne une valeur entière.

Déclaration

La déclaration d'une fonction se place au début du programme avant la fonction « `main` » (qui est elle même une déclaration de fonction). Elle se réalise comme suit :

```
type_retour nom_fonction( type1 var1, type2 var2,
                                type3 var3...)


{
     [ corps de la fonction ]
}
```

« `type_retour` » défini le type de la valeur de sortie de la fonction (en langage C, les fonctions n'ont, en sortie, qu'une et une seule valeur). Si une fonction ne retourne pas de valeur, le « `type_retour` » est « `void` », qui veut dire néant en anglais :

```
void bonjour( )


{
     printf( "bonjour le monde!\n");
}
```

Ensuite, on déclare le nom de la fonction et les paramètres d'entrées. Pour chaque paramètre d'entrée, on faut spécifier son type et son nom. A l'intérieur du corps de la fonction, un paramètre peut être considéré comme une variable locale à la fonction.

La valeur retournée par la fonction est déterminée par le mot clef « `return` » suivit de la valeur. Lors que le programme arrive à ce mot clef, il quitte immédiatement la fonction. Il peut donc y avoir plusieurs mot clef « `return` » à l'intérieur du corps d'une fonction (le premier rencontré provoque la sortie de la fonction), et ils ne sont pas forcèment placé à la fin de celle-ci. Dans notre exemple, c'est la valeur de la variable local « `resultat` » qui est retournée.

Utilisation

L'appel à une fonction se fait comme suit :

```
nom_fonction( exp1, exp2, exp3...)
```

En C, le passage de paramètres se fait « par valeur ». A l'appel de la fonction, on donne en argument des expressions (constantes, variables ou expressions complexe) du type défini à la déclaration.

L'ensemble de cet appel, est lui même une expression dont le type est celui défini par le type de retour de la fonction.

Les variables définies à l'intérieur du corps d'une fonction (idem pour la fonction « `main` ») ainsi que les paramètres définis lors de la déclaration d'une fonction sont locales, c'est à dire qu'ils ne peuvent être utilisés que dans le corps de cette fonction. Dans notre premier exemple, on ne peut par exemple utiliser la variable « `operation` » définie dans la fonction « `main` » dans le corps de la fonction « `addition` ».

Exercice 1

Ecrire un programme qui définit une fonction « `factorielle` » prenant en entrée un entier et en sortie un autre entier. Utiliser cette fonction dans le corps de la fonction « `main` », demander à l'utilisateur de rentrer un entier et calculer sa factorielle.

Exercice 2

Ecrire un programme qui définit une fonction « `min` » prenant en entrée trois flottants et retournant le plus petit d'entre eux (le type de retour est donc lui aussi un flottant). Utiliser cette fonction dans le corps de la fonction « `main` ».

Variables locales/globales

Nous avons vu que les variables définies à l'intérieur du corps d'une fonction sont locales à cette fonction, c'est à dire qu'elles ne sont visibles et utilisables qu'à l'intérieur de celle-ci. Le langage C permet de définir des variables globales, qui sont utilisables et visibles dans tous les programme. Pour cela il suffit de les déclarer avant les fonctions, de la manière suivante :

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



/* declaration des variables globales */
int un_mille = 1852;
int tableau[3][3];
double pi = 3.1415926;

int conversion_en_metre( int valeur_en_mille)


{
     return valeur_en_mille * un_mille;
}
```

```
int main(int argc, char **argv)


{
      ...
      printf( "pi vaut : %d\n", pi);
      ...
}
```

Exercice 3

Nous allons écrire un programme permettant de trouver les entiers premiers par le crible d'Erathostène. Déclarer dans un premier temps en variable globale un tableau de 10 000 entiers. Dans ce tableau, l'élément d'indice 0 représente le chiffre 0, l'élément d'indice 1, le chiffre 1, etc.

1. Ecrire une fonction mettant à 1 tous les éléments de ce tableau,
2. Ecrire une fonction retournant l'indice du premier élément non nul de ce tableau à partir de l'indice 2, si il n'y en a pas, retourner -1,
3. Ecrire une fonction qui pour un nombre, met à 0 tous les éléments dont l'indice est multiple de ce nombre. Soit pour 3, les éléments d'indices 3, 6, 9, 12, etc. Faire attention à ne pas dépasser la taille maximum du tableau.

Pour réaliser le crible d'Erathostène, on utilise toutes les fonctions définies ci-dessus. Dans un premier temps, mettre à un tous les éléments du tableau (fonction 1). Ensuite, rechercher à partir de l'indice 2, le premier nombre non nul (fonction 2), ce nombre est premier donc on l'affiche. Mettre ensuite à zéro dans le tableau tous les éléments dont l'indice est multiple ce nombre non nul (fonction 3). Recommencer la recherche de nombre non nul, jusqu'à ce que l'on obtienne -1.