|  |  |  |
| --- | --- | --- |
|  | **INSTITUT SUPERIEUR DES ETUDES TECHNOLOGIQUES DE NABEUL** | |
|  | Travaux Pratiques N° 5 : Les fonctions | |
| **Classe : TI1** | | **Enseignante : Sana REFAI** |

## 1. Objectifs

Dans cet atelier, nous allons essayer de manipuler les fonctions en C.

## 2. Introduction

Le langage C permet de découper un programme en plusieurs parties appelées modules.

Un module est un ensemble de données et d’instructions qui fournissent une solution à une petite partie d’un problème plus complexe.

Quelques avantages de la modularité :

* Meilleure lisibilité
* Diminution du risque d’erreurs
* Possibilité de tests sélectifs
* Réutilisation de modules déjà existants
* Simplicité d’entretien
* ...

En langage C, la structuration des programmes en modules se fait à l’aide des fonctions et main() est une de ces fonctions.

L'imbrication des fonctions n'est pas autorisée en C : une fonction ne peut pas être définie à l'intérieur d'une autre fonction. Par contre, une fonction peut appeler une autre fonction. Cette dernière doit être déclarée avant celle qui l'appelle.

Une fonction possède un et un seul point d'entrée, mais éventuellement plusieurs points de sortie (à l'aide du mot return).

Une variable connue uniquement d'une fonction est une variable locale. Une variable connue de tout le programme est une variable globale.

## 3. Définition d’une fonction

Une fonction en langage C est définie, **syntaxiquement**, comme suit :

Syntaxe:

**<Type Résultat> <Nom Fonction> (<TypePar1> <NomPar1>, <TypePar2> <NomPar2>, ...)**

**{**

**<déclarations locales >**

**<suite d’instructions>**

**}**

**Type résultat Nom fonction Type Pramètre1 Nom Paramètre1**

Exemple:

**short max (short A, short B)**

**{**

**/\* déclarations locales \*/**

**short maximum ;**

**/\* suite d’instructions \*/**

**if (A > B) maximum = A;**

**else maximum = B;**

**/\* renvoi du résultat \*/**

**return (maximum);**

**}**

**REMARQUES**

* Les noms des paramètres et de la fonction sont des identificateurs qui correspondent aux mêmes restrictions définies pour les identificateurs des variables
* Si la fonction n’a pas de paramètres, déclarer la liste des paramètres comme **void** ou ne rien mettre entre les ( )

**int main ( )**

**{**

**...**

**}**

**ou**

**int main (void)**

**{**

**...**

**}**

* Quand une fonction ne fournit pas de résultat, indiquer comme void le type du résultat

Exemple:

**void Bonjour ()**

**{**

**printf ("Bonjour\n");**

**}**

* Une fonction peut fournir comme résultat:
  + un type arithmétique (entier ou rationnel)
  + une structure
  + un pointeur
  + void
  + Le type par défaut est le type int

**int main ( )**

**{...}**

**ou**

**main ( )**

**{...}**

* Le renvoi du résultat :Pour toute fonction renvoyant un résultat, il faut fournir le résultat à l’aide de la commande :

**return**

**<expression>;**

**double Tan (double x)**

Exemple:

**{**

**if (cos (x) != 0)**

**return (sin (x) / cos (x));**

**else**

**return (-1);**

**}**

## 4. Déclaration d’une fonction ( Prototype)

**Syntaxe:**

**<Type Résultat> <Nom Fonction> (<TypeP1> [<NomP1>], <TypeP2> [<NomP2>], ...);**

Syntaxe:

Toute fonction doit être déclarée avant d’être utilisée (sauf la fonction main, on peut ne pas la déclarer)

* **Règles pour la déclaration des fonctions**
* ***Déclaration locale :*** avant la déclaration des variables,
* la fonction est déclarée localement dans la fonction qui l’appelle, elle est disponible uniquement pour celle-ci.

Exemple:

**void Rectangle (int lg, int ht) /\* Fonction Rectangle \*/**

**{ void Ligne (int lg);**

**int i;**

**for (i=0; i < ht; i++)**

**Ligne (lg);**

**}**

**void Ligne (int lg); /\* Fonction Ligne \*/**

**{ for (int i=0; i < lg; i++)**

**{putchar (‘ \* ’);}**

**putchar (‘ \n ’);**

**}**

* ***Déclaration globale :*** après les **#include**
* , la fonction est déclarée globalement au début du fichier. Elle est disponible pour toutes les fonctions du programme (la plus simple et la plus sûre quand le programme est complexe).

**#include <stdio.h>**

Exemple:

**void Rectangle (int lg, int ht );**

**void Ligne (int lg);**

**void main ( )**

**{**

**int L, H;**

**printf (“Donner la longueur :\n “);**

**scanf (“%d“, &L);**

**printf (“Donner la hauteur :\n “);**

**scanf (“%d“, &H);**

**Rectangle (L, H);**

**}**

**/\* Fonction Rectangle \*/**

**void Rectangle (int lg, int ht)**

**{**

**int i;**

**for (i=0; i < ht; i++)**

**Ligne (lg);**

**}/\* Fonction Ligne \*/**

**void Ligne (int lg);**

**{**

**int i;**

**for (i=0; i < lg; i++)**

**{putchar (‘ \* ’);}**

**putchar (‘ \n ’);**

**}**

* ***Déclaration implicite par la définition :*** la fonction est automatiquement disponible pour toutes les fonctions qui suivent sa définition.
* La fonction principale **main** n'a pas besoin d'être déclarée
* Lors de la *déclaration*, le nombre et le type des paramètres doivent nécessairement correspondre à ceux de la *définition* de la fonction.

## 5. Variable locales et globales

Il existe deux types de variables : **variables locales** et **variables globales**.

* **Variables locales**

Les variables déclarées dans une fonction sont uniquement visibles à l’intérieur de cette fonction. On dit que ce sont des **variables** **locales** pour cette fonction (valable pour toutes les fonctions y compris main).

Exemple:

**void Ligne (int lg); /\* Fonction Ligne \*/**

**{**

**int i; /\* i est une variable locale \*/**

**for (i=0; i < lg; i++)**

**{putchar (‘ \* ’);}**

**putchar (‘ \n ’);**

**}**

* **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 des variables **globales**

**#include <stdio.h>**

Exemple:

**int A; /\* A est une variable globale \*/**

**void main ( )**

**{**

**void fct1 ( );**

**... if (A > 0) ...**

**}**

**void fct1 ( )**

**{**

**... A ++; …**

**}**

## 6. Paramètres formels et effectifs

Les paramètres se trouvant dans l’en-tête d’une fonction sont appelés des  **paramètres formels**. Ils permettent au sein du corps de la fonction de décrire ce qu’elle doit faire.

**int max (int x, int y);**

Exemple:

**{**

**if (x > y)**

**return (x);**

**else**

**return (y);**

**}**

Les paramètres fournis à l’appel de la fonction se nomment des **paramètres effectifs.** Ces paramètres vont remplacer les paramètres formels lors de l’exécution du programme.

**void main ( )**

Exemple:

**{**

**int C, A, B ;**

**A = 3;**

**B = 7;**

**C = max (A, B);**

**...**

**}**

## 7. Passage de paramètres

Il existe deux types de passage de paramètres : **passage par valeur** et **passage par adresse (référence)**.

### 7.1 Passage par valeur

Les fonctions n’obtiennent que les valeurs de leurs paramètres et n’ont pas d’accès aux variables.

A l’intérieur d’une fonction, on peut changer les valeurs des paramètres formels sans modifier les valeurs originales dans les fonctions appelantes.

**void main ( )**

Exemple:

**{ void permuter (int A, int B) ;**

**int x, y;**

**x = 3;**

**y = 4;**

**permuter (x, y);**

**/\* x = 3 et y = 4, x et y ne changent pas de valeurs \*/**

**...**

**}**

**void permuter (int A, int B) /\* A = 3 et B = 4 \*/**

**{ int aux;**

**aux = A;**

**A = B;**

**B = aux; /\* A = 4 et B = 3 \*/**

**}**

### 7.2 Passage par adresse (réserver pour le semestre 2)

Pour changer la valeur d’une variable de la fonction appelante :

* La fonction appelante doit fournir l’adresse de la variable
* La fonction appelée doit déclarer le paramètre correspondant comme pointeur, la variable est donc atteinte à l’aide d’un pointeur

**void main ( )**

Exemple:

**{ void permuter (int \*A, int \*B) ;**

**int x, y;**

**x = 3;**

**y = 4;**

**permuter (&x, &y);**

**/\* x = 4 et y = 3, x et y changent de valeurs \*/**

**...**

**}**

**void permuter (int \*A, int \*B) /\* (\*A) = 3 et (\*B) = 4 \*/**

**{**  **int aux;**

**aux = \*A;**

**\*A = \*B;**

**\*B = aux; /\* (\*A )= 4 et (\*B) = 3 \*/**

**}**

### 7.3 Passage de l'adresse d'un tableau à une dimension

* Méthode :

Comme il est impossible de passer 'la valeur' de tout un tableau à une fonction, on fournit ***l'adresse d'un élément du tableau***.

En général, on fournit l'adresse du premier élément du tableau, qui est donnée par ***le nom du tableau***.

* Déclaration :

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)**

Exemple:

**{**

**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[]**

Dans la suite, nous allons utiliser la première notation pour mettre en évidence que le paramètre est un *pointeur* *variable* que nous pouvons modifier à l'intérieur de la fonction.

* 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:

Exemple:

**strlen(CH)** **/\* résultat: 9 \*/**

**strlen(P)** **/\* résultat: 9 \*/**

**strlen(&CH[4])** **/\* résultat: 5 \*/**

**strlen(P+2)** **/\* résultat: 7 \*/**

**strlen(CH+2)** **/\* résultat: 7 \*/**

Dans les trois dernières appels, nous voyons qu'il est possible de fournir *une partie* d'un tableau à une fonction, en utilisant l'adresse d'un élément à l'intérieur de tableau comme paramètre.

* Remarque pratique :

Pour qu'une fonction puisse travailler correctement avec un tableau qui n'est pas du type **char**, il faut aussi fournir la dimension du tableau ou le nombre d'éléments à traiter comme paramètre, sinon la fonction risque de sortir du domaine du tableau.

**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.

Exemple:

**void LIRE\_TAB(int N, int \*PTAB)**

**{**

**printf("Entrez %d valeurs : \n", N);**

**while(N)**

**{**

**scanf("%d", PTAB++);**

**N--**

**}**

**}**

## 8. Les Exercices

### 8.1 Exercice 1

Ecrire une fonction qui permet de calculer le factoriel d’un entier n. faire l’appel au niveau du main

### 8.2 Exercice 2

On va simuler le fonctionnement d’une calculatrice en développant les 4 fonctions basique addition, soustraction, multiplication et division. Dans le main vous faits le saisie d 2 entiers a et b et l’opération à effectuer. Donner le résultat en utilisant les fonctions déjà définies.

### 8.3 Exercice 3

Créer un ensemble de procédures et de fonctions permettant de manipuler facilement les heures et les minutes et composé de :

1. La fonction « Minutes », qui calcule le nombre des minutes correspondant à un nombre d’heures et un nombre de minutes donnés.
2. La fonction ou la procédure « HeuresMinutes » qui réalise la transformation inverse de la fonction Minute.
3. La procédure « AjoutTemps » qui additionne deux couples de données heures et minutes en utilisant les deux fonctions précédentes.

### 8.4 Exercice 4

 Ecrire une fonction Apparition qui prend en paramètres une chaîne de caractères et une lettre et calcule le nombre d’apparition de la lettre dans cette chaîne.

8.5 Exercice 5

 1. Ecrire une fonction Position qui prend en paramètres un tableau d’entiers et un entier et retourne la position de cet entier dans le tableau. S’il n’existe pas, cette fonction doit retourner la valeur -1.

2. Ecrire une fonction Supprimer qui prend en paramètres un tableau et un indice k et supprime l’élément de position k du tableau.

Dans la fonction principale main() vous devez utiliser les deux fonctions Position  et Supprimer pour supprimer un élément d’un tableau d’entiers autant de fois qu’il existe.

8.6 Exercice 6

1. Ecrire une fonction LIRE\_TAB qui permet de remplir un tableau TAB de dimension maximale NMAX et de dimension effective N, donnés comme paramètres.

2. Ecrire une fonction ECRIRE\_TAB qui permet d’afficher un tableau TAB de dimension N. 3. Ecrire une fonction SOMME\_TAB qui retourne la somme des éléments du tableau TAB, de dimension N.

4. Ecrire un programme principal qui permet de tester ces différentes fonctions sur un tableau d’entiers T de taille maximale 100, et de taille effective L donné par l’utilisateur.

### 8.7 Exercice 7: Mon premier jeux en C

Ecrire un programme C modulaire qui simule le jeu d’allumettes. Le joueur et l’ordinateur partagent un tas de 21 allumettes. A tour de rôle, ils doivent retirer 1, 2 ou 3 allumettes du tas ; celui qui retire la dernière allumette a perdu. Votre programme doit s’assurer de la validité du nombre d’allumettes à retirer qui doit être compris entre 1 et le minimum de 3 et du nombre d’allumettes du tas. A chaque tirage, il faut afficher l’état du jeu, c’est-à-dire le nombre d’allumettes restantes. A la fin de la partie, le programme doit afficher le joueur gagnant. N.B. Afin de réaliser le choix de l’ordinateur, utiliser l’instruction n= random(max +1); qui permet de générer un nombre aléatoire compris 0 et max