– Algorithmique – TP N°5 – PROCEDURES & FONCTIONS –

I– Introduction : Les **procédures et fonctions** sont utilisées pour **simplifier** l'écriture des programmes.

Le programme global est ainsi **décomposé** en plusieurs parties faciles à écrire, à comprendre et à faire évoluer à l'inverse d'un programme de grande taille écrit en un seul bloc.

Le programme (**principal**) décrit alors la méthode générale de résolution du problème traité et les procédures et fonctions traitent les détails.

Cette décomposition permet aussi de **réutiliser** les procédures et fonctions au lieu de les réécrire à chaque fois car souvent les mêmes traitements sont effectués plusieurs fois dans le même programme.

Résumé : ① En langage C, tous les traitements sont effectués par des fonctions. ② Les procédures sont un cas particulier de fonctions. ③ Le programme principal est la fonction ***int main( )***.

II– Les fonctions

✓ Une fonction retourne un seul résultat avec l'instruction **return**. Elle est définie selon la syntaxe :

typeResultat nomFonction(TypeParametre1 parametre1, … , TypeParametreN parametreN)

{ /* Déclarations de variables locales */

/* Liste des instructions*/

return resultat ; }

typeResultat : est l'un des types ENTIER (char, short, int, long, …) , REEL (float, double, …), CARACTERE (char) ou ENREGISTREMENT (struct).

Exemple : Une fonction qui calcule le **maximum** de deux **entiers n1** et **n2** peut être écrite comme suit :

```
int maximum(int n1, int n2) { int max ; // max est une variable locale
    if(n1 > n2) max = n1 ;
    else max = n2 ;
    return max ; }
```

✓ L'instruction **return** permet de retourner le résultat et terminer la fonction (ou la procédure). Si l'instruction **return** est rencontrée, toutes les instructions qui la suivent ne seront pas exécutées.

Exemple : La fonction **maximum** précédente peut être réécrite comme suit :

```
int maximum(int n1, int n2){ if(n1 > n2) return n1 ;
                             return n2 ; }
```

✓ Une fonction est, généralement, utilisée (appelée) dans une expression arithmétique ou logique. Elle peut aussi servir comme paramètre (passé par valeur) à une autre fonction ou procédure.

Exemples d'appels : ① int n = n3 + maximum(n1, n2) ; ② Pour trouver le maximum de trois entiers, on peut écrire : int max = maximum(maximum(n1, n2), n3) ; // Le 1er paramètre est un appel de fonction.

III – Les procédures

✓ Une procédure est une fonction dont le type de retour est **void** (vide). Il signifie que la procédure ne retourne aucune valeur. **Syntaxe de déclaration** : **void** nomProcedure(Liste_des_paramètres)

✓ La procédure est utilisée (appelée) comme une instruction. Elle ne peut pas être utilisée dans une expression arithmétique ou logique, ni passée en paramètre à une autre procédure ou fonction.

✓ Dans une procédure, l'utilisation de l'instruction **return** reste possible, mais elle n'est pas suivie d'un résultat (**return ;**).

Remarque 1 : Les instructions **scanf** et **printf** sont, en réalité, deux fonctions. ① **scanf** : En plus des valeurs lues à partir du clavier, elle retourne le nombre de variables correctement lues à partir du clavier.

② **prinf** : En plus de l'affichage sur écran, cette fonction retourne le nombre de caractères affichés.

Remarque 2 : Si une fonction est appelée comme une procédure (comme une instruction), la fonction sera exécutée mais la valeur de retour n'est pas exploitée (utilisée). **Exemples** : scanf et printf.

IV – Passage de paramètres par valeur (par copie) :

Dans ce type de passage de paramètres, la valeur du paramètre est copiée dans une variable locale sans toucher la valeur d'origine. C'est cette variable locale qui est utilisée dans la fonction (ou la procédure) appelée. Aucune modification de la variable locale dans la fonction ne modifie la variable passée en paramètre, parce que ces modifications ne s'appliquent qu'à une copie de cette dernière.

Exemple : La fonction **facto** qui calcule la factorielle d'un entier positif **n** peut être écrite comme suit :

```
int facto(int n) { int f = 1 ; for( ; n > 1 ; n--) f *= n ; return f ;}
```

Explication : Malgré que la fonction **facto** modifie le paramètre **n** en le décrémentant (**n--**), il retrouvera sa valeur initiale à la fin de la fonction car cette dernière travaille sur une copie de **n** (**n** est passé par valeur).

✓ Un paramètre passé par valeur (par copie) peut recevoir une valeur constante, une variable, une expression ou même un appel à une fonction. **Exemples** : facto(5), facto(x), facto(n1 + n2), facto(maximum(n1, n2)).

V – Passage de paramètres par adresse (par référence) :

Dans ce type, il n'y a pas de copie de la valeur du paramètre mais la copie de son adresse. La procédure (ou fonction) travaille directement avec la variable passée en paramètre, elle peut ainsi modifier sa valeur. Toute modification du paramètre dans la procédure appelée entraîne la modification de la variable passée en paramètre.

✓ Un paramètre passé par adresse à une procédure (**rarement** à une fonction) est précédé par le symbole *.

Exemple : int *a ; La variable **a** est dite un **pointeur** sur un **int**. (***a)** est la **valeur pointée** par le pointeur **a**.

Une procédure qui incrémente la valeur d'un **int a** peut être écrite comme suit :

```
void incrementer(int *a) { (*a)++ ; /* ou *a = *a + 1 ; */ }
```

Exemple d'appel de la procédure : int n = 10 ; incrementer(**&**n) ; printf("%d", n) ; // n aura la valeur 11.

A l'appel, il faut utiliser le symbole **&** (lu adresse) comme déjà utilisé dans scanf.

VI – Variables locales et variables globales

✓ Une **variable globale** est déclarée à l'extérieur du corps de toute fonction, et peut donc être utilisée dans n'importe quelle procédure ou fonction (y compris la fonction **main**). En général, les variables globales sont déclarées au début du programme, immédiatement après les directives **#include** et **#define**.

✓ Une **variable locale** ne peut être utilisée que dans la fonction ou le bloc ({ …bloc… }) où elle est définie.

Remarques : ① Si une variable globale n'est pas initialisée par l'utilisateur, elle est initialisée par défaut à zéro. ② Si une variable locale n'est pas initialisée par l'utilisateur, sa valeur est indéterminée. **Attention** !

VII – LES EXERCICES – LES PROCEDURES ET LES FONCTIONS

Exercice 1 : Tables de multiplication

```
7 × 1 =  7
7 × 2 = 14
7 × 3 = 21
7 × 4 = 28
7 × 5 = 35
7 × 6 = 42
7 × 7 = 49
7 × 8 = 56
7 × 9 = 63
7 × 10 = 70
```

Q1) Ecrire une procédure ***afficherMultiples*** qui permet d'afficher la table de multiplication d'un entier **n** compris entre **1** et **10** (**1 ≤ n ≤ 10**). En face, la table de multiplication de **n = 7.**

Q2) Ecrire le programme principal (la fonction ***int main( )***) qui permet de lire un entier ***n*** compris entre 1 et 10 et d'afficher sa table de multiplication.

Q3) Modifier le programme principal pour afficher toutes les tables de multiplication de 1 à 10.

Exercice 2 : Affichage d'étoiles

Pour chaque procédure, écrire un programme C qui permet de la tester.

Q1) Ecrire une procédure ***ligne(n)*** qui permet d'afficher une ligne de **n** étoiles '*' (astérisques).

Exemple : ********** est une ligne de 10 étoiles.

Q2) En utilisant la procédure ***ligne***, écrire une procédure ***rectangle(n, m)*** qui permet d'afficher un rectangle d'étoile de ***m*** lignes de **n** étoiles. **Exemple** : Ci-dessous un rectangle de **3** lignes de **20** étoiles.

```
********************
********************
********************
```

Q3) En utilisant la procédure ***rectangle***, écrire une procédure ***carre(n)*** qui permet d'afficher un carré d'étoiles de **n** lignes de **n** étoiles. Ci-dessous un carré de 4 lignes de 4 étoiles.

```
****
****
****
****
```

Exercice 3 : Ordre croissant

Q1) Pour deux entiers ***a*** et ***b***, écrire une procédure ***Trier2*** qui permute ***a*** et ***b***, si nécessaire, pour que l'état de sortie soit ***a ≤ b*** (***a*** et ***b*** seront dans l'ordre croissant).

Q2) En utilisant la procédure ***Trier2***, écrire une procédure ***Trier3*** sur 3 entiers ***(a, b et c)*** qui permet d'avoir les trois variables ***a, b et c*** dans l'ordre croissant ***(a ≤ b ≤ c)***.

Q3) Ecrire un programme C qui permet de lire 3 entiers (***x***, ***y***, ***z***) puis les afficher dans l'ordre croissant.

Exercice 4 : Maximum/Minimum

Q1) Ecrire une fonction ***max2(x, y)*** qui retourne le plus grand de deux nombres entiers ***x*** et ***y***.

Q2) Utiliser la fonction ***max2*** pour écrire une deuxième fonction ***max3(x, y, z)*** qui calcule le plus grand de trois nombres entiers ***x***, ***y*** et ***z***.

Q3) Ecrire un programme C qui permet de lire trois entiers ***a***, ***b***, ***c*** et d'afficher leur maximum.

Q4)** Adapter les fonctions précédentes ainsi que le programme principal pour afficher le **minimum** de trois entiers ***a, ***b*** et ***c.***

*** Exercice 5 : Nombres Premiers**

Un nombre est dit premier s'il n'admet que deux diviseurs : 1 et lui-même.

Q1) Ecrire une fonction ***Premier(n)*** qui permet de vérifier si un entier ***n*** est premier ou pas.

Q2) Ecrire une procédure qui permet d'afficher les ***m*** premiers nombres premiers.

Exemple : Si **m = 8**, la procédure doit afficher : 2 3 5 7 11 13 17 19.

Q3) Ecrire un programme C qui permet de lire un entier ***m*** et d'afficher les ***m*** premiers nombres premiers.

Exercice 6 : Fonctions itératives ou récursives ?

Pour chaque fonction, écrire un programme C qui permet de la tester.

Q1) ***Factorielle*** : Ecrire une fonction ***Facto(n)*** qui permet de calculer ***n !* = 1 * 2 * 3 * …* n**

Q2) ***Puissance*** : Ecrire une fonction ***Puiss(x, n)*** qui renvoie $x^n = x * x * \dots * x$ (n fois)

Q3) ***Factorielle*** : Ecrire une fonction récursive ***FactoRec(n)*** qui permet de calculer ***n !*** en sachant que :

n ! = n * (n-1) ! et **0 ! = 1 ! = 1**

Q4) ***Puissance*** : Ecrire une fonction récursive ***PuissRec(x, n)*** qui permet de calculer x^n en utilisant le fait que :

$$x^n = \begin{cases} 1 & Si\ n = 0 \\ x^{n-1} \times x & Si\ n > 0 \end{cases}$$

Q5) ***Puissance*** : Ecrire une fonction récursive ***PuissRec2(x, n)*** qui permet de calculer x^n en utilisant le fait

que : $x^n = \begin{cases} 1 & Si\ n = 0 \\ x^{n/2} \times x^{n/2} & Si\ n > 0\ et\ n\ pair \\ x^{(n-1)/2} \times x^{(n-1)/2} \times x & Si\ n > 0\ et\ n\ impair \end{cases}$

*** Q6)** L'algorithme d'Euclide permettant de calculer le **PGCD** (Plus Grand Commun Diviseur) de deux entiers strictement positifs ***A*** et ***B*** tels que ***A > B*** est défini comme suit :

$$PGCD(A,B) = \begin{cases} PGCD(B,\ A \bmod B) & Si\ B \neq 0 \\ A & Si\ B = 0 \end{cases}$$

Ecrire une fonction récursive permettant de déterminer le ***PGCD*** de ***A*** et ***B***.

*** Q7)** ***Suite de Fibonacci*** : Ecrire une fonction récursive ***Fibo(n)*** permettant de calculer le n^{ieme} terme de la

suite de ***Fibonacci*** définie par : $U_n = \begin{cases} 0 & Si\ n = 0 \\ 1 & Si\ n = 1 \\ U_{n-1} + U_{n-2} & Si\ n \geq 2 \end{cases}$

Q8) Pour un entier positif **n (n ≥ 0)**, écrire une fonction itérative puis une autre récursive qui permet :

- Calculer le nombre des chiffres de **n**.
- Calculer la somme des chiffres de **n**.
- Calculer le produit (la multiplication) des chiffres de **n**.
- * Vérifier si un chiffre ($0 \leq$ chiffre ≤ 9) existe dans le nombre **n**.
- * Calculer le nombre d'occurrences d'un chiffre ($0 \leq$ chiffre ≤ 9) dans le nombre **n**.

*** : Travail facultatif**