TP 2 : Fonctions et procédures simples

Objectifs :

- Maîtriser l'écriture de fonctions.
- Etre capable d'écrire en C un programme structuré en plusieurs fonctions.

Stéphanie CHOLLET – Ioannis PARISSIS
El Mehdi KHALFI – Ayoub SOURY

2015/2016

Fonctions et procédures

1. Notions de base

Lire les sections 1.15.1 à 1.15.3 du polycopié.

Exemple de définition d'une fonction :

```
/* la fonction somme retourne un int */
/* ses paramètres formels sont les int i et j */
int somme(int i, int j) {
     int res; /* variable locale */
     res = i + j;
     return(res); /* retour de la valeur de res */
}
```

`int somme(int i, int j)` est le *prototype* de la fonction `somme`.

Appel de fonction :

```
#include <stdio.h>

int main(){
     int r, a, b;

     a = 3;
     b = 2;
     r = somme(a, b);
     printf("a + b = %d\n", r);
}
```

En C, il n'existe pas de procédure, à proprement parler. En fait, une procédure est une fonction qui ne retourne pas de valeur (auquel cas, on spécifie `void` comme type de retour) :

```
#include <stdio.h>

void aff_somme(int i, int j) {
     int res = i + j;
     printf("somme = %d\n", res);
}

int main(){
     int r, a, b;

     a = 3;
     b = 2;
     aff_somme(a, b);
}
```

Exercices

Pour chacun de ces exercices, on écrira, **dans un même fichier**, la ou les fonctions demandées ainsi que la fonction `main` les appelant.

Exercice 1 : Fibonacci

On rappelle la suite de Fibonacci définie par :

$$u_0 = 0$$
$$u_1 = 1$$
$$u_n = u_{n-1} + u_{n-2} \text{ si } n > 1$$

- Ecrire une fonction `fibonacci` calculant le terme de rang n de la suite dont le prototype est : `int fibonnacci(int n);`
- Ecrire ensuite une fonction `main` demandant la valeur de n à l'utilisateur et affichant le terme correspondant de la suite.
- Testez votre programme avec différentes valeurs de n en justifiant leur choix (commentaire dans votre fichier).

Exercice 2 : PGCD

On rappelle que le pgcd est défini par les relations suivantes (a et b étant des entiers naturels) :

pgcd(a, 0) = a

pgcd(a, b) = pgcd(b, r) avec r = a mod b, si b ≠ 0 (mod est le reste de la division entière).

- Ecrire une fonction (**utilisant une itération**) `pgcd`, à deux paramètres entiers, retournant le pgcd de ses paramètres.
- Ecrire une fonction `main` demandant deux valeurs entières à l'utilisateur et affichant leur pgcd.
- Testez votre programme avec différents entiers en justifiant leur choix (commentaire dans votre fichier).

Exercice 3 : Factorielles

- Ecrire une fonction `factorielle` qui calcule et retourne la valeur de n! (1 x 2 x 3 x…x n).
- Ecrire une fonction `factorielleBis` à un paramètre entier m qui calcule et retourne la valeur du plus petit entier positif n tel que n! (factorielle de n) soit supérieur à m.
- Ecrire ensuite une fonction `main` demandant la valeur de n à l'utilisateur et affichant le résultat de chaque fonction ci-dessus.
- Testez votre programme avec différentes valeurs de n en justifiant leur choix (commentaire dans votre fichier).

Exercice 4 : Jeu de multiplication

On veut écrire une procédure `jeuMulti` qui demande à l'utilisateur de réciter sa table de multiplication. L'utilisateur commence par entrer un nombre entre 2 et 9 (si le nombre est incorrect, le programme redemande). Ensuite l'algorithme affiche une à une les lignes de la table de multiplication de ce nombre, en laissant le résultat vide et en attendant que l'utilisateur entre le résultat. Si celui-ci est correct, on passe à la ligne suivante, sinon on affiche un message d'erreur donnant la bonne valeur et on termine. Si toutes les réponses sont correctes, on affiche un message de félicitations. On représente ci-dessous une exécution possible (les entrées de l'utilisateur sont affichées en italiques) :

Valeur de n : *12*
Réessayez : la valeur doit être comprise entre 2 et 9
Valeur de n : *6*
1 x 6 = *6*
2 x 6 = *12*
3 x 6 = *21*
Erreur ! 3 x 6 = 18 et non 21

1. Ecrire la procédure `jeuMulti`.
2. Ecrire une nouvelle procédure `jeuMultiPoints` qui ne s'arrête pas quand une réponse fausse est donnée, mais affiche à la fin le nombre d'erreurs commises et un message éventuel de félicitations.
3. Testez ces deux procédures en justifiant vos tests.