J1MI2013: Algorithmes et Programmes: feuille 2

Itérer avec la boucle for

Travaux dirigés

Exercice 1. En utilisant la boucle `for`, écrire une fonction `sommeCarres(n)` qui calcule et renvoie $\sum_{i=1}^{n} i^2$.

Exercice 2. On considère le programme :

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

int
main(void){

  for (int i=1 ; i<=10 ; i=i+1){
    printf("%d ", i);
  }
  printf("\n");

  return EXIT_SUCCESS;
}
```

Quel est le résultat de son exécution ?
Que faut il modifier pour obtenir un programme qui :

- affiche dans l'ordre croissant les nombres entiers strictements compris entre 10 et 30.
- affiche dans l'ordre décroissant les nombres entiers strictements compris entre 10 et 30.
- affiche dans l'ordre croissant les nombres naturels pairs inférieurs à 100.

Exercice 3. On veut écrire une fonction qui calcule 2^n en utilisant la boucle `for`. Pour cela compléter le code suivant :

```
int
puissanceDe2(int n){
  int p2 = 1;

  /* completer en utilisant l'instruction for */
  return p2;
}
```

Exercice 4. En utilisant la boucle `for`, écrire une fonction `affichePuissance2` qui affiche les puissances de 2 de 1024 à 2.

Exercice 5. Factorielle itérative
On sait que `n!` est défini par :

$$\begin{cases} 0! = 1! = 1 \\ \forall n > 1, n! = n * (n-1) * (n-2) * \ldots * 2 * 1 \end{cases}$$

En utilisant la boucle `for`, écrire une fonction `factorielleIterative` utilisant cette définition.

Exercice 6. Écrire une fonction `void afficheTableMult(int n)` qui affiche une table de multiplication de taille n.
Par exemple pour n = 6, on doit obtenir l'affichage :

```
1   2   3   4   5   6
2   4   6   8  10  12
3   6   9  12  15  18
4   8  12  16  20  24
5  10  15  20  25  30
6  12  18  24  30  36
```

Exercice 7. Suite de Fibonacci
En utilisant la boucle `for`, écrire une fonction `fibonacci` qui calcule et renvoie le n-ième terme de la suite de Fibonacci ($n \geq 0$) définie par :

$$\begin{cases} u_0 = u_1 = 1 \\ \forall n \geq 2, u_n = u_{n-1} + u_{n-2} \end{cases}$$

Exercice complémentaire 1. Écrire une fonction `facteurImpair`, qui, étant donné un entier naturel `n` non-nul, renvoie le plus grand diviseur impair de `n`. Le résultat sera obtenu par une succession de divisions de `n` par 2.