TP N°3
Les fonctions

1. Punition 1

Écrire une fonction *punition1* qui affiche 10 fois "Je dois apprendre mes cours".

Tester la fonction dans le main

2. Punition 2

Écrire une fonction *punition2* qui affiche n fois "Je dois apprendre mes cours".

Tester la fonction dans le main

3. Aire d'un rectangle

Écrire une fonction *aireRectangle* prenant 2 paramètres flottants (longueur et largeur) et permettant de retourner l'aire d'un rectangle.

Tester la fonction dans le main sur des valeurs de votre choix.

4. Les triangles

Un triangle est :

- Équilatéral si tous ses cotés sont de même longueur.
- Isocèle si deux coté ont la même longueur.
- Rectangle si ses trois cotés a, b et c vérifient : $a^2 + b^2 = c^2$
- Quelconque s'il ne vérifie aucune de ces propriétés.

Écrire quatre fonctions *EstEquilateral*, *EstIsocele*, *EstRectangle* et *EstQuelconque* qui renvoie 1 si la réponse est oui, 0 sinon. Le programme doit avoir le fonctionnement suivant (en rouge les saisies de l'utilisateur) :

```
./TP3EX4
Tiangles :
----------
Entrez la longueur des trois cotés du triangle ? 4 3 5
Le triangle est rectangle.
```

5. Opérations entières

Écrire un programme qui demande à l'utilisateur deux entiers et qui affiche le menu suivant (en rouge les saisies de l'utilisateur) :

```
./TP3EX5
Operations entieres :
---------------------
Entrez deux entiers ? 8 5
Menu :
      1. Addition
      2. Soustraction
      3. Multiplication
      4. Division
      5. Quitter
Choix ? 2
La soustraction 8 - 5 vaut : 3
```

Vous écrirez une fonction pour chacune des opérations.

6. Factorielle

Écrire la fonction *factorielleIterative* prenant en entier en paramètre et permettant de retourner la valeur de sa factorielle.

Écrire la fonction récursive *factorielleRecursive* prenant en entier en paramètre et permettant de retourner la valeur de sa factorielle.

Tester la fonction dans le main sur des valeurs de votre choix.

7. Suite de Fibonacci

La suite de Fibonacci est une suite de nombres dont chaque terme est la somme des deux précédents. Les deux premiers termes de la suite sont par définition égaux à 1. La suite est définie comme suit :

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2} \quad \text{pour} \quad n > 2 \end{cases}$$

Écrire la fonction *FibonacciIterative* permettant de calculer le nième terme de la suite.

Tester la fonction dans le main sur des valeurs de votre choix.

Écrire la fonction *FibonacciRecursive* permettant de calculer le nième terme de la suite par récurrence

Le programme doit avoir le fonctionnement suivant (en rouge les saisies de l'utilisateur) :

```
./TP3EX7
Suite de Fibonacci :
--------------------
Entrez un nombre positif (negatif pour finir) > 0
Entrez un nombre positif (negatif pour finir) > 1
Le 1 er terme de la suite de Fibonacci est : 1
Entrez un nombre positif (negatif pour finir) > 4
Le 4 ieme terme de la suite de Fibonacci est : 3
Entrez un nombre positif (negatif pour finir) > 8
Le 8 ieme terme de la suite de Fibonacci est : 21
Entrez un nombre positif (negatif pour finir) > -1
```

8. Puissance d'un nombre

Écrire une fonction *PuissanceIterative* qui calcule x^n pour *x* flottant quelconque et *n* entier positif ou nul. Sa déclaration est la suivante :

```
float PuissanceIterative (float x, int n);
```

Écrire une nouvelle fonction *PuissanceRecursive* en mettant à profit la remarque suivante :

$$x^n = \begin{cases} x \times x^{n-1} & \text{si n est impair} \\ (x^2)^{\frac{n}{2}} & \text{si n est pair} \end{cases}$$

Sa déclaration est la suivante :

```
float PuissanceRecursive (float x, int n);
```

Le programme doit avoir le fonctionnement suivant (en rouge les saisies de l'utilisateur) :

```
./TP3EX8
Puissance :
-----------
Entrez un réel et un entier positif (negatif pour finir) > 3.0 2
3.0 a la puissance 2 vaut : 9.00
Entrez un réel et un entier positif (negatif pour finir) > 5.0 3
5.0 a la puissance 3 vaut : 125.00
Entrez un réel et un entier positif (negatif pour finir) > 8.0 5
8.0 a la puissance 5 vaut : 32768.00
Entrez un réel et un entier positif (negatif pour finir) > -1 -1
```

9. Binôme de Newton

Les coefficients C_n^p du binôme de Newton :

$$(x+y)^n = \sum_{p=0}^{n} C_n^p x^{n-p} y^p$$

Ils peuvent être calculés récursivement grâce à la formule du triangle de Pascal :

$$\begin{cases} \forall n, n \in N, C_n^0 = C_n^n = 1 \\ \forall n, n \in N, n \geq 2, \forall p, p \in N, 0 < p < n, C_n^p = C_{n-1}^{p-1} + C_{n-1}^p \end{cases}$$

Écrire une fonction récursive *CoefficientBinomeRecursive* qui calculera les coefficients du binôme de Newton. Pour le calcul de la puissance d'un nombre, vous pourrez utiliser la fonction de l'exercice précédent. La déclaration de la fonction est la suivante :

```
int CoefficientBinomeRecursive (int n, int p);
```

Ecrivez une fonction itérative *CoefficientBinomeIterative* sachant que :

$$C_n^p = \frac{n!}{(p! * (n-p)!)}$$

Vous pourrez écrire une fonction *Factorielle*. La déclaration des fonctions sont les suivantes :

```
int Factorielle (int n);
int CoefficientBinomeIterative (int n, int p);
```

Écrire un programme principal affichant le résultat du calcul du binôme de Newton pour les valeurs x, y et n entrées au clavier. Le programme doit avoir le fonctionnement suivant (en rouge les saisies de l'utilisateur) :

```
./TP3EX9
Binome de Newton :
------------------
Entrez trois entiers positifs (negatif pour finir) ? 2 3 4
(2 + 3)^4 : 625
Entrez trois entiers positifs (negatif pour finir) ? -1 -1 -1
```