Exercice Fonction

Exercice 1

Faire une fonction `sqrt` elle aura pour Prototype :

```
int sqrt(int nbrToSqrt)
```

Cette fonction prendra un nombre et le multipliera par lui-même.

Exercice 2

Faire une fonction `getNextPow2` qui prendra un nombre et qui renverra la puissance de 2 supérieur ou égale au nombre rentré.
Prototype : `int getNextPow2(int nbr)`
(Exemple : 5 -> 8, 8 -> 8, 9 -> 16, 1026 -> 2048)

Ps : pensez a réutiliser les fonctions déjà écrite.

Exercice 3

Faire une fonction `getPreviousPow2` qui prendra un nombre et qui renverra la puissance de 2 inférieur ou égale au nombre rentré.
Prototype : `int getPreviousPow2(int nbr)`
(Exemple : 5 -> 4, 8 -> 8, 9 -> 8, 1026 -> 1024)

Ps : pensez a réutiliser la fonctions `getNextPow2`.

Exercice 4

Faire une fonction `PowOf2` qui prendra un nombre et renverra le facteur de 2 qui le compose.
Prototype : `int PowOf2(int nbr)`
(Exemple : 5 -> 2, 8 -> 3, 9 -> 3, 1026 -> 10)
(Explication Exemple : 5 -> 2^2, 8 -> 2^3, 9 -> 2^3, 1026 -> 2^10)

Ps : pensez a réutiliser les fonctions déjà écrite.

Exercice 5

Écrivez une fonction appelée `ConvertToBinary` qui prend en paramètre un entier n et qui retourne dans le terminal le nombre en binaire représentant la valeur binaire de n.
Prototype : `void ConvertToBinary(int nbr)`

Exemple : 5 -> 101, 8 -> 1000, 9 -> 1001, 1026 -> 10000000010

Ps : un nombre binaire est un nombre en base 2. Il est composé de 0 et de 1.
Pss: utilisez la fonction `PowOf2` pour trouver le facteur de 2 qui compose le nombre.

Exercice 6

Faire une fonction qui parcourt un tableau a deux dimensions et qui affiche le contenu de celui-ci.
Prototype : `void draw2DimTab(int [,] tab)`

Exercice 6 tips

Un tableau a deux dimension se déclare comme ceci :

```
int [,] tab = new int[2,3];
```

Il on peut l'instancier comme ceci :

```
int [,] tab = new int[2,3] { {1,2,3}, {4,5,6} };
```

Exercice 6 tips

Pour connaître la longueur d'une dimensions d'un tableau, on peut utiliser la méthode `GetLength(index de la dimentions)`
Exemple : `tab.GetLength(0)` renverra la longueur de la première dimensions du tableau : `2`.

Exercice 7

Écrivez une fonction appelée `GenerateBoard` qui prend en paramètre deux entiers `rows` et `columns` et qui retourne un tableau à deux dimensions de string.
Représentant une grille de jeu avec rows nombre de lignes et columns nombre de colonnes.
Votre fonction devra utiliser des boucle `for` imbriquée pour remplir la grille de "." (point).
Prototype : `string [,] GenerateBoard(int rows, int columns)`

Exercice 7

Exemple :

```
GenerateBoard(2,3) ->
[
    [" · ", " · ", " · "],
    [" · ", " · ", " · "]
]
```

Exercice 8

Écrivez une fonction appelée `PlaceToken` qui prend en paramètre un tableau à deux dimensions de string, un entier row, un entier column et une string token.
Elle modifie la grille en plaçant token à la position spécifiée par row et column.

Exercice 9

Écrivez une fonction appelée `CheckPosToken` qui prend en paramètre un tableau a deux dimensions de string, un entier row, un entier column.
Elle vérifie si la position spécifiée par row et column est existe dans la tableau a deux dimensions en argument.
Et si c'est le cas vérifie si la position est égale a "." (point).
Elle renverra true si la position est valide et false sinon.

Prototype :

```
bool CheckPosToken(string [,] tab, int row, int column)
```

Exercice 10

Faire une fonction `checkWin` qui prendra en paramètre un tableau a deux dimensions de string, un int row, un int column.
Elle vérifiera si a la position indique par row et column, il y a 3 jetons alignes(ligne, colonne ou diagonal).
Prototype : `bool checkWin(string [,] tab, int row, int column)`

Exercice 11

Faire une fonction `StartMorpion` qui ne prendra pas de paramètre et qui renverra rien.
Elle lancera une partie de morpion.
Elle attendra la saisie de l'utilisateur pour placer un jeton.
Elle alternera les joueurs jusqu'a ce qu'un des deux gagne ou qu'il n'y ait plus de place disponible.
Le joueur 1 jouera avec le jeton "X" et le joueur 2 jouera avec le jeton "O".
Elle affichera draw si il n'y a pas de gagnant.
Elle affichera win pour le joueur gagnant.

Exercice 12

Faire une fonction `draw` qui prend 3 arguments et renvoie rien.

```
void draw(int height, int length, int form)
```

- 📌 form sera comprit entre 0 et 3 (inclut)
- 📌 height et length corresponde a réciproquement la hauteur et la longueur de la forme a afficher

L'objectif de cette fonction est de dessiner des carrés,
Si `form` est impaire, le carré sera plein sinon vide.

Exercice 12

Si form est égale a 0 ou 1, la fonction dessinera pour:

```
draw(2,2,1)
o--o
|oo|
|oo|
o--o
```

```
draw(1,1,0)
o-o
| |
o-o
```

```
draw(4, 2, 0)
o--o
|  |
|  |
|  |
|  |
o--o
```

Exercice 12

Si form est égale a 2 ou 3, la fonction dessinera pour:

```
draw(2,2,3)      draw(1,1,2)          draw(1, 5, 2)
uxxu             uxu                  uxxxxxu
xuux             x x                  x     x
xuux             uxu                  uxxxxxu
uxxu
```

Exercice 13

Créer un dictionnaire de `string, int` nommer phoneBook.

Créer deux fonction qui permettent d’afficher et d'ajouter des contacts.

Pour cela, créer une boucle infini qui demandera a l'utilisateur s'il veut :

- 📌 ajouter un nouveau contact, ce qui appellera votre fonction addContacts qui prendra un nom, et un numero de téléphone.
- 📌 afficher ses contacts, ce qui appellera votre fonction listContacts, qui affichera votre dictionnaire.
- 📌 exit, qui arrêtera votre programme.

Évidement, un numero de téléphone sera compose uniquement de 10 chiffres, a vous de vous en assurer. Et un même numero ne pourra pas être attribué a deux personnes.

Exercice 14

Rajouter a votre programme la possibilité de supprimer un contact.

Pour cela, ajouter une fonction deleteContact qui prendra en paramètre un nom et supprimera le contact correspondant.