EXERCICE 06 – ITÉRATIVES 1

1 Travail à effectuer

Ouvrez le projet fourni avec cet énoncé. Il devrait déjà contenir un fichier `Program.cs`.

1.1 Affichage de nombres

Localisez la fonction `PrintNumbers`. Dans cette fonction, écrivez le code nécessaire pour afficher à la console tous les nombres contenus entre un nombre minimum et un nombre maximum (tous les deux inclusivement).

1.2 Somme d'une suite de nombres

Localisez la fonction `ComputeSum`. Dans cette fonction, écrivez le code effectuant la somme de tous les nombres de 1 jusqu'à une certaine limite X (inclusivement).

Vérifiez que votre code fonctionne avec ces tests.

#	Variables	
	Entrées	Sortie
1	`limitNumber = 1`	`sum = 1`
2	`limitNumber = 2`	`sum = 3`
3	`limitNumber = 5`	`sum = 15`

Cette question peut aisément être répondue à l'aide d'une répétitive `for` ou d'une répétitive `while`. Produisez le code pour les deux répétitives et laissez-en une en commentaires.

1.3 Produit d'une suite de nombres pairs

Localisez la fonction `ComputeProductOfEvenNumbers`. Dans cette fonction, écrivez le code effectuant le produit (la multiplication) de tous les nombres pairs de 2 jusqu'à une certaine limite X (inclusivement). Pour simplifier la question, vous pouvez supposer que le nombre limite sera toujours pair et plus grand ou égal à 2.

Vérifiez que votre code fonctionne avec ces tests.

#	Variables	
	Entrées	Sortie
1	`limitNumber = 2`	`product = 2`
2	`limitNumber = 6`	`product = 48`
3	`limitNumber = 10`	`product = 3840`

Cette question peut aisément être répondue à l'aide d'une répétitive `for` ou d'une répétitive `while`. Produisez le code pour les deux répétitives et laissez-en une en commentaires.

1.4 Somme des chiffres qui composent un nombre.

Localisez la fonction `ComputeSumOfDigits`. Dans cette fonction, écrivez un algorithme permettant de calculer la somme des chiffres qui composent un nombre. Par exemple, la somme des nombres de 15 est 6 (soit $1 + 5 = 6$).

Vérifiez que votre code fonctionne avec ces tests.

#	Variables	
	Entrées	Sortie
1	`number = 0`	`sumOfDigits = 0`
2	`number = 2`	`sumOfDigits = 2`
3	`number = 15`	`sumOfDigits = 6`
4	`number = 48`	`sumOfDigits = 12`

Indice

La division de deux nombres entiers est aussi entière (sans reste). Par exemple, $148 \div 10 = 14$.
Aussi, n'oubliez pas l'opérateur modulo (`%`). Par exemple, $148 \,\%\, 10 = 8$.
Par conséquent,

- Si vous prenez le modulo par 10 de votre nombre initial, vous obtenez les unités.
- Si vous prenez la division par 10 vous obtenez le nombre de dizaines.

Il ne vous reste qu'à additionner le nombre d'unités ET à recommencer avec le nombre de dizaines tant que ce nombre n'est pas zéro.

1.5 [Bonus #2] Plus grand commun diviseur

Localisez la fonction `GreatestCommonDivisor`. Dans cette fonction, écrivez un algorithme permettant de calculer le plus grand commun diviseur ($PGCD$) entre deux nombres. Pour ce faire, utilisez l'algorithme d'Euclide, qui va comme suit :

PGCD(a,b) = PGCD(b, a%b) tant que b!=0.

où l'opérateur % est l'opérateur modulo qui calcule le reste de la division entière.

Comme toujours, vérifiez que votre code fonctionne avec ces tests.

#	Variables	
	Entrées	Sortie
1	`a = 3` `b = 3`	`pgcd = 3`
2	`a = 3` `b = 6`	`pgcd = 3`
3	`a = 6` `b = 3`	`pgcd = 3`
4	`a = 27` `b = 12`	`pgcd = 3`
5	`a = 128` `b = 78`	`pgcd = 8`

Cette question est plus difficile à comprendre qu'à coder… Prenez une feuille de papier et essayer de tracer l'algorithme. Par exemple,

a	b	PGCD(a,b)=
128	72	PGCD(72, 128%72)=PGCD(72,56)
72	56	PGCD(56, 72%56)=PGCD(56,16)
56	16	PGCD(16, 56%16)=PGCD(16,8)
16	8	PGCD(8, 16%8)=PGCD(8,0)
8	0	FIN. PGCD(128,72)=8

2 Modalités de remise

Remettez votre projet Visual Studio sur LÉA, dans la section travaux, à l'intérieur d'une archive *Zip*. Supprimez tous les dossiers temporaires, à savoir les dossiers `.vs`, `TestResults`, `bin` et `obj`.