TP C#8 : TinyBistro.

1 Introduction

1.1 Quésaco ?

La Bistromathique est l'un des grands projets de l'EPITA que vous aurez à réaliser plus tard durant votre scolarité. Cette dernière est censée calculer des nombres potentiellement infinis et en base quelconque.

Si vous n'avez jamais entendu parler de "The Hitchhiker's Guide to the Galaxy" nous vous conseillons vivement de prendre un peu de temps pour votre culture et de remédier a cela car vous en aurez besoin en temps qu'Epitéen(ne).

1.2 Objectif

Bien sûr, nous n'allons pas vous demander de refaire entièrement la Bistro mais seulement une petite partie. A la fin de ce projet, votre programme devra être capable de lire un fichier contenant deux nombres (strictement positifs et potentiellement infinis) et une unique opération.

Pour vous simplifier la vie (et parce que nous sommes de bons Samaritains) vous n'aurez que des nombres positifs à gérer et si une soustraction sauvage apparaît, le nombre de gauche sera toujours supérieur au nombre de droite (n'y voyez aucun avis politique).

Notez aussi que l'optimisation de votre Bistro ne sera prise en compte qu'en tant que bonus. Le plus important est d'avoir quelque chose qui marche. S'il court, c'est un bonus.

1.3 Rendu

```
login_x.zip
    |– login_x/
        |– AUTHORS
        |– README
        |– TinyBistro/
            |– TinyBistro.sln
            |– TinyBistro/
                |– Tout sauf bin/ et obj/
```

2 Cours

Maintenant que vous maîtrisez les notions basiques (et principales) du C#, nous ne devrions plus avoir à vous expliquer les principes basiques mais plutôt une infinité de petites choses par ci par là. Mais surtout par là (et par vous même).

Parmi ces petites choses : la surcharge d'opérateur.

Imaginons que vous vouliez redéfinir l'opérateur `+` pour une *`Certaine`* classe. Nous vous conseillerons vivement de reprendre le code suivant :

```
1 public static Certaine operator+(Certaine a, Certaine b)
2     {
3         /*
4          * Do what you want
5          * I don't want to know
6         */
7         return new Certaine(/* I don't know */);
8     }
```

Une fois cet opérateur surchargé, il sera possible d'écrire :

```
1 Certaine a = new Certaine(/*...*/);
2 Certaine b = new Certaine(/*...*/);
3 Certaine c = a + b;
```

Ce qui n'était pas possible avant.

3 Let's code

Avant tout chose pensez à récupérer le squelette fourni avec le TP et à bien lire tout le sujet avant de vous lancer.

3.1 BigNum

Dans cette première partie vous allez créer la classe représentant votre nombre potentiellement infini. Celle ci se compose d'un unique attribut et d'un certains nombre de méthodes.

Pour stocker votre nombre, vous ne pourrez pas simplement utiliser un uint64 car ce dernier ne peut dépasser "18 446 744 073 709 551 615" et nous voulons pouvoir faire plus qu'un nombre a 20 chiffres (c'est dommage hein !). Du coup vous allez devoir stocker votre nombre dans une liste.

Pour cela, votre liste devra contenir à l'index 0 son plus petit chiffre. Plus visuellement : 18 446 sera représenté par la liste [6][4][4][8][1]. On prendra aussi comme convention que le nombre 0 sera représenté par la liste vide [ ].

Il ne vous reste plus qu'à implémenter les fonctions suivantes :

Constructeur

```
1 public BigNum(string source)
```

Il doit prendre en paramètre une string représentant votre nombre. Ce dernier doit remplir votre liste à partir de la string en paramètre. Attention, le nombre représenté par la string 1234 doit être stocké dans la liste comme ceci : [4][3][2][1].

GetNumDigits

```
1 public int GetNumDigits()
```

Cette méthode ne prend pas d'argument et retourne la longueur du BigNum, c'est a dire le nombre de chiffres qui le composent.

AddDigit

```
1 public void AddDigit(int Digit)
```

Cette fonction prend en paramètre un chiffre et l'ajoute en fin de la liste représentant le BigNum.

GetDigit

```
1 public int GetDigit(int position)
```

Cette méthode prend une position (un entier) en argument et permet de récupérer le chiffre de la liste à cette position. Si la position est supérieure ou égale au nombre de chiffres dans le BigNum, vous devez renvoyer une exception de type `OverflowException`. Notez que `GetDigit(0)` appelée sur le BigNum représentant `123` renvoie `3`.

SetDigit

```
1 public void SetDigit(int digit, int position)
```

Cette méthode prend un entier (entre 0 et 9) en argument ainsi qu'une position et permet de changer la valeur du ième chiffre de la liste. Si jamais la position du chiffre à ajouter est plus grande que le nombre de chiffres dans le BigNum, rajoutez autant de zéros que nécessaire. Faites bien attention à retirer les premiers chiffres de la liste qui sont des zéros à la fin de cette fonction.

Print

```
1 public void Print()
```

Cette méthode ne prend pas de paramètre et affiche dans la console le BigNum de manière lisible par un humain. Bien sur, vous n'oublierez pas de gérer le cas du zéro. Cette fonction doit afficher un retour à la ligne après le BigNum.

3.2 Parsing

Maintenant que vous avez de quoi implémenter votre BigNum, vous avez besoin de récupérer la string contenant ce dernier. Pour cela, vous allez tout d'abord devoir récupérer le contenu du ficher ou est écrit le calcul. Puis découper ce que vous avez obtenu pour en extraire les 2 BigNum ainsi que l'opérateur.

Nous vous demandons donc d'implémenter une classe `Parser`. Pour plus de simplicité, cette classe sera `static`.

Pour cette classe, vous êtes libres de l'implémentation. Votre seule contrainte est qu'elle doit contenir une méthode `Parse`. Cette dernière une fois appelée doit afficher dans la console le résultat du calcul contenu dans le fichier `operation.txt` (ce fichier doit se trouver au même niveau que l'exécutable).

Petit conseil : allez y par étape : récupérer le 1er BigNum, ensuite l'opérateur puis le second BigNum. Ensuite vous n'avez plus qu'à appliquer l'opération qui correspond grâce a la partie suivante.

3.3 BigNum again

Maintenant que vous avez récupéré l'opération et les nombres, il ne vous reste plus qu'à faire le calcul.

Pour cela vous allez devoir surcharger les opérateurs de la classe BigNum comme le montre la partie cours. Les opérateur à surcharger sont : `==, !=, <, >, +, -, *, / et %`

En tant que fin limier que vous êtes, vous avez du apercevoir les fonctions `Equals` et `GetHashCode`. Ne vous en préoccupez pas, elle servent juste a enlever des warning ou erreurs de compilation.

Aujourd'hui vous allez apprendre une petite leçon de vie. L'un des grands aspect de la programmation vient de la recherche. Tout ne vous sera pas toujours expliqué, il faut apprendre à chercher. Saluez chaleureusement vos amis Google et Wikipedia. Et bien sur tout copier-coller sera sanctionné.

C'est pour cela que nous allons vous donner des noms d'algo de base pour les opérateurs et non pas du code pré mâché. Bien sur vous êtes vivement encouragés à utiliser des algo plus performants si vous le souhaitez (une fois que ceux de base fonctionnent). Cela vous rapportera évidement des points bonus (et n'oubliez pas de le préciser dans le README !). Si vous avez besoin de pistes pour de meilleurs algos, n'hésitez pas a demander.

`==, !=, > et <` devraient sembler évidents. Attention cependant à la manière dont vous avez stocké vos BigNum.

`+` : Bon là pas de nom d'algo, l'addition vue en primaire suffit. Pensez juste à bien gérer les retenues si une case de la liste dépasse 10.

`-` : Pareil que pour le `+` pas grand chose a dire a part de revoir vos cours de primaire et de bien gérer le dépassement et la retenue. On rappel que le nombre de gauche sera toujours supérieur au nombre de droite.

`*` : La ça se complique un peu. On vous conseille encore une fois de vous tourner vers la multiplication vue en primaire. Si les choses faciles c'est pas assez pour vous et que vous voulez du challenge, jetez un oeil du côté de l'algorithme de Karatsuba. Si le challenge c'est trop facile pour vous et que vous avez envie de faire palir de jalousie vos camarades et faire rougir de joie vos ACDC, vous pouvez implémenter la transformation de Fourier rapide (FFT).

`/` : Toujours pareil, décidément la primaire c'était super ! Challenge : The D algorithm.

`%` : Si vous avez implémenté correctement la division, cette fonction ne devrait pas vous prendre plus de 10 secondes (réellement !).

ATTENTION : Ce n'est pas parce que ces méthodes de calculs ont été vues étant petit que leur implémentation est triviale. N'oubliez pas que vous avez des assistants pour vous aider, profitez-en !

Brace yourselves, deadline is coming ...