Références et Tableaux

Consignes

Dans ce TP, vous ne devez rendre **uniquement** que les fichiers sources (`.cs`) de vos programmes.
La structure du rendu doit être comme ceci :

```
/rendu-tp-login_x.zip
   |-- login_x /
          |-- Matrix.cs
          |-- Murray.cs
          |-- References.cs
          |-- AUTHORS
          |-- README
```

`rendu-tp-login_x.zip`	nom de l'archive à rendre
`AUTHORS`	vous devez remplacer le contenu du fichier par une étoile `*`, un espace, puis votre login (ex : `login_x`) suivi d'un retour à la ligne
`README`	vous devez écrire dans ce fichier, tout commentaire sur le TP, votre travail ou bien plus généralement sur vos points forts/faibles. un fichier `README` vide sera considéré comme une archive sale (malus).

Il va de soi que toutes les occurrences de `login_x` sont à remplacer par **votre** login.

Tout le TP est à réaliser en mode console : `Fichier` ⇒ `Nouveau` ⇒ `Projet` ⇒ `Application Console`

TESTEZ VOTRE CODE !

In computer science, an array type is a data type that is meant to describe a collection of elements (values or variables), each selected by one or more indices (identifying keys) that can be computed at run time by the program. Such a collection is usually called an array variable, array value, or simply array. By analogy with the mathematical concepts of vector and matrix, array types with one and two indices are often called vector type and matrix type, respectively.

– **wikipedia**

1 Tableaux

Fichier à rendre : `Murray.cs`

Dans cette section nous allons aborder les tableaux [1], il va de soi que les routines doivent être en impératif. Vous devez respecter les prototypes donnés.

bases

Écrire une fonction qui initialise le contenu du tableau passé en paramètre, de façon aléatoire[1].

```
static void ArrayInit(int[] v) { /* FIXME */ }
```

Écrire une fonction qui affiche le contenu d'un tableau.

```
static void ArrayPrint(int[] v) { /* FIXME */ }
```

Écrire une fonction qui calcule la moyenne des valeurs contenues dans le tableau.

```
static double Average(int[] v) { /* FIXME */ }
```

Écrire une fonction qui inverse l'ordre des valeurs d'un tableau.

```
static void ArrayReverse(int[] v) { /* FIXME */ }
```

tris

Écrire les fonctions suivantes triant [2] notre tableau en ordre croissant :

En utilisant un tri à bulles.

```
static void ArraySortBubble(int[] v) { /* FIXME */ }
```

(bonus) En utilisant un tri par insertion.

```
static void ArraySortInsert(int[] v) { /* FIXME */ }
```

(bonus) En utilisant un tri par sélection.

```
static void ArraySortSelect(int[] v) { /* FIXME */ }
```

1. cf. annexes
2. `http://en.wikipedia.org/wiki/Sorting_algorithm`

The **`ref`** *keyword causes an argument to be passed by reference, not by value. The effect of passing by reference is that any change to the parameter in the method is reflected in the underlying argument variable in the calling method. The value of a reference parameter is always the same as the value of the underlying argument variable.*

Do not confuse the concept of passing by reference with the concept of reference types. The two concepts are not the same. A method parameter can be modified by **`ref`** *regardless of whether it is a value type or a reference type. There is no boxing of a value type when it is passed by reference.*

To use a **`ref`** *parameter, both the method definition and the calling method must explicitly use the* **`ref`** *keyword.*

– **msdn.microsoft.com**

2 Références

Fichier à rendre : `References.cs`

Nous allons dans cette partie manipuler le passage par références (équivalent des paramètres globaux).

fonctions

Écrire une procédure qui échange le contenu de deux entiers.

```
1 static void Swap(ref int a, ref int b) { /* FIXME */ }
```

Écrire une procédure qui "range" les entiers `a`,`b`,`c`, du plus grand au plus petit dans les paramètres.

```
1 static void MaMeMi(ref int a, ref int b, ref int c) { /* FIXME */ }
```

Écrire une procédure qui rectifie un temps donné, en répartissant l'excédent correctement.

```
1 static void TimeCorrecter(ref int weeks, ref int days,
2                            ref int hours, ref int minutes,
3                            ref int seconds) { /* FIXME */ }
```

In mathematics, a matrix (plural matrices) is a rectangular array of numbers, symbols, or expressions, arranged in rows and columns. The individual items in a matrix are called its elements or entries.
– wikipedia

3 Matrices

Fichier à rendre : `Matrix.cs`

Dans le but de se préparer pour le projet de S2#, nous allons voir les matrices qui peuvent vous simplifier [3] la vie.

opérations

Écrire une fonction qui calcule, si possible, la trace d'une matrice. La fonction retournera `true` si le calcul est possible et stockera le résultat dans la variable adéquate (non initialisée), `false` sinon.

```
static bool MatrixTrace(double[,] a, ref double b);
```

Écrire une fonction qui additionne, si possible, deux matrices. La fonction retournera `true` si le calcul est possible et stockera le résultat dans une troisième (non initialisée), `false` sinon.

```
static bool MatrixAdd(double[,] a, double[,] b, ref double[,] c);
```

Écrire une procédure qui multiplie une matrice par un scalaire. La procédure stockera le résultat dans la deuxième matrice à notre disposition.

```
static void MatrixScalar(double a, double[,] b, ref double[,] c);
```

Écrire une procédure qui calcule la transposée d'une matrice. La procédure stockera le résultat dans la deuxième matrice à notre disposition.

```
static void MatrixTranspose(double[,] a, double[,] b);
```

(bonus) Écrire une fonction qui multiplie, si possible, deux matrices. La fonction retournera `true` si le calcul est possible et stockera le résultat dans une troisième (non initialisée), `false` sinon.

```
static bool MatrixMult(double[,] a, double[,] b, ref double[,] c);
```

3. note du créateur du sujet : *lire complexifier*

Annexes

tableaux

```
int[] t1;                           // an unitialised integer array
int[] t2 = new int[5];              // an integer array (length: 5)
int[] t3 = { 21, 12, 0 };           // an integer array (length: 3)

t1 = t3;                            // t1 and t3 are now the same array
t1[2] = 13;                         // change the 3rd element of t1
                                    // try to print t3 now

t2.Length;                          // length attribute (= 5)
```

```
int[,] m1 = new int[4,5];           // a two-dimensional array (4x4)
m1.Length;                          // length attribute (= 20)
m1.GetLength(0);                    // 1st dimension length (= 4)
m1.GetLength(1);                    // 2nd dimension length (= 5)
```

```
int[][] m2 = new int[2][3];         // error

int[][] m3 = new int[2][];          // initialize the 1st dimension
for (int i = 0; i < 2; ++i)
    m3[i] = new int[3];             // initialize the 2nd dimension
                                    // could be different on each row

m3.Length;                          // length attribute (= 2)
m3[0].Length;                       // length attribute (= 3)
```

```
int[,,,,,,,,,,,]  d;                // ???
```

```
int[,] tab = new int[4][4];         // error
```

aléatoire

```
Random g = new Random()             // instanciate a new generator

g.Next(10);                         // return an integer [0..10[
g.Next(5, 20);                      // return an integer [5..20[
```

références

```
class RefExample {
    static void Increment(int a) { a += 5; }
    static void RefIncrement(ref int a) { a += 5; }

    static void Main(string[] args) {
        int b = 10;
        Increment(b);
        Console.WriteLine(b);     // should display 10
        RefIncrement(ref b);
        Console.WriteLine(b);     // should display 15
        Console.ReadKey();
    }
}
```

```
class ReffArray {
    static void Init(int[] tab) {
         for (int i = 0; i < tab.Length; ++i)
             tab[i] = i * i;
         tab = new int[5];
    }

    static void RefInit(ref int[] tab) {
         for (int i = 0; i < tab.Length; ++i)
             tab[i] = i * i;
         tab = new int[5];
    }

    static void PrintTab(int[] tab) { /* */ }

    static void Main(string[] args) {
        int[] tab = new int[8];

        for (int i = 0; i < tab.Length; ++i)
            tab[i] = i;

        PrintTab(tab);

        Init(tab);
        PrintTab(tab);

        RefInit(ref tab);
        PrintTab(tab);
    }
}
```