

Exercices

Algorithmes simples avec



Calcul de la valeur absolue d'un nombre réel	p. 4
Résolution de l'équation du second degré dans R	p. 6
Calcul des nombres de Armstrong	p. 8
Calcul de nombres parfaits	p. 10
Calcul du pgcd de 2 entiers (méthode Euclide)	p. 12
Calcul du pgcd de 2 entiers (méthode Egyptienne)	p. 14
Calcul de nombres premiers (boucles while et do...while)	p. 16
Calcul de nombres premiers (boucles for)	p. 18
Calcul du nombre d'or	p. 20
Conjecture de Goldbach	p. 22
Méthodes d'opérations sur 8 bits	p. 24
Chaînes palindromes 2versions	p. 28
Convertir une date numérique en lettres	p. 31
Convertir un nombre écrit en chiffres romains	p. 33
Tri à bulles tableau d'entiers	p. 34
Tri par insertion tableau d'entiers	p. 36
Recherche linéaire dans un tableau non trié	p. 39
Recherche linéaire dans un tableau déjà trié	p. 43
Recherche dichotomique dans un tableau déjà trié	p. 46

*Dans la majorité des exemples de traduction d'algorithmes simples nous reprenons volontairement des exemples déjà traités dans un autre livre pdf, les **fondements de Java**, le lecteur remarquera la similarité du code C# et du code Java pour le même exercice.*

Pour le lecteur familiarisé avec Java, lorsqu'il y a dissemblance du code C# avec le code Java, nous avons fourni des explications spécifiques au code C#.

Classes, objets et IHM avec

Problème de la référence circulaire.....	p. 49
Classe de salariés dans une entreprise fictive	p. 51
Classe de salariés dans un fichier de l'entreprise	p. 62
Construction d'un ensemble de caractères	p. 70
Construction d'un ensemble générique	p. 76
Construction d'une IHM de jeu de puzzle	p. 84

Les exercices utilisent des classes spécifiques au langage C#, si le lecteur veut traduire ces exemples en code Java ou en code Delphi, il doit soit chercher dans les packages Java ou Delphi des classes possédant les mêmes fonctionnalités soit les construire lui-même.

Algorithme

Calcul de la valeur absolue d'un nombre réel

Objectif : Ecrire un programme C# servant à calculer la valeur absolue d'un nombre réel x à partir de la définition de la valeur absolue. La valeur absolue du nombre réel x est le nombre réel $|x|$:

$$|x| = x, \text{ si } x \geq 0$$

$$|x| = -x \text{ si } x < 0$$

Spécifications de l'algorithme :

```
lire( x );  
si x ≥ 0 alors écrire( '|x| =', x)  
sinon écrire( '|x| =', -x)  
fsi
```

Implantation en C#

Ecrivez avec les deux instructions différentes "if...else.." et "...?.. : ...", le programme C# complet correspondant à l'affichage ci-dessous :

```
Entrez un nombre x = -45  
|x| = 45
```

Proposition de squelette de classe C# à implanter :

```
class ApplicationValAbsolue {  
    static void Main(string[] args) {  
        .....  
    }  
}
```

La méthode **Main** calcule et affiche la valeur absolue.

Classe C# solution

Une classe C# solution du problème avec un **if...else** :

```
using System;
namespace CsExosAlgo1 {
class ApplicationValAbsolue1 {
    static void Main(string[] args) {
        double x;
        System.Console.WriteLine("Entrez un nombre x = ");
        x = Double.Parse( System.Console.ReadLine( ) );
        if (x<0) System.Console.WriteLine("|x| = "+(-x));
        else System.Console.WriteLine("|x| = "+x);
    }
}
```

Explication sur l'instruction :

```
x = Double.Parse ( System.Console.ReadLine( ) );
```

Le gestionnaire d'entrée sortie C# à partir de la classe Console renvoie à travers la méthode ReadLine() une valeur saisie au clavier de type string. Il est donc obligatoire si l'on veut récupérer un nombre réel au clavier (ici **double** x;) de transtyper le réel tapé correctement sous forme de chaîne au clavier, et de le convertir en un réel de type **double** ici, grâce à la méthode Parse de la classe enveloppe **Double** du type **double**.

Remarquons que cette version simple ne protège pas des erreurs de saisie. Pour être plus robuste le programme devrait intercepter l'exception levée par une éventuelle erreur de saisie signalée par une exception du type **FormatException** :

```
try {
    x = Double.Parse( System.Console.ReadLine( ) );
}
catch ( FormatException ) { ///...traitement de l'erreur de saisie }
```

Une classe C# solution du problème avec un **"... ? ... : ..."** :

```
using System;
namespace CsExosAlgo1
{
class ApplicationValAbsolue2 {
    static void Main(string[] args) {
        double x;
        System.Console.WriteLine("Entrez un nombre x = ");
        x = Double.Parse( System.Console.ReadLine( ) );
        System.Console.WriteLine("|x| = "+ (x < 0 ? -x : x) );
    }
}
}
```

Algorithme

Algorithme de résolution de l'équation du second degré dans R.

Objectif : On souhaite écrire un programme C# de résolution dans R de l'équation du second degré : $Ax^2 + Bx + C = 0$

Il s'agit ici d'un algorithme très classique provenant du cours de mathématique des classes du secondaire. L'exercice consiste essentiellement en la traduction immédiate

Spécifications de l'algorithme :

Algorithme Equation

Entrée: A, B, C ∈ Réels

Sortie: X1, X2 ∈ Réels

Local: Δ ∈ Réels

début

lire(A, B, C);

Si A=0 alors début{A=0}

Si B = 0 alors

Si C = 0 alors

écrire(R est solution)

Sinon{C ≠ 0}

écrire(pas de solution)

Fsi

Sinon {B ≠ 0}

X1 ← C/B;

écrire (X1)

Fsi

fin

Sinon {A ≠ 0} début

$\Delta \leftarrow B^2 - 4*A*C$;

Si $\Delta < 0$ alors

écrire(pas de solution)

Sinon { $\Delta \geq 0$ }

Si $\Delta = 0$ alors

X1 ← $-B/(2*A)$;

écrire (X1)

Sinon{ $\Delta \neq 0$ }

X1 ← $(-B + \sqrt{\Delta})/(2*A)$;

X2 ← $(-B - \sqrt{\Delta})/(2*A)$;

écrire(X1 , X2)

Fsi

Fsi

fin

Fsi

FinEquation

Implantation en C#

Ecrivez le programme C# qui est la traduction immédiate de cet algorithme dans le corps de la méthode Main.

Proposition de squelette de classe C# à implanter :

```
class ApplicationEqua2 {  
    static void Main(string[] args) {  
        .....  
    }  
}
```

Conseil :

On utilisera la méthode **static** Sqrt(double x) de la classe **Math** pour calculer la racine carrée d'un nombre réel :

$\sqrt{\Delta}$ se traduira alors par : **Math.Sqrt(delta)**

Classe C# solution

```
using System;
namespace CsExosAlgo1
{
    class ApplicationEqua2 {
        static void Main (string[] arg) {
            double a, b, c, delta ;
            double x, x1, x2 ;
            System.Console.Write("Entrer une valeur pour a : ");
            a = Double.Parse( System.Console.ReadLine() );
            System.Console.Write("Entrer une valeur pour b : ");
            b = Double.Parse( System.Console.ReadLine() );
            System.Console.Write("Entrer une valeur pour c : ");
            c = Double.Parse( System.Console.ReadLine() );
            if (a == 0) {
                if (b == 0) {
                    if (c == 0) {
                        System.Console.WriteLine("tout reel est solution");
                    }
                    else { // c ≠ 0
                        System.Console.WriteLine("il n'y a pas de solution");
                    }
                }
                else { // b ≠ 0
                    x = -c/b ;
                    System.Console.WriteLine("la solution est " + x);
                }
            }
            else { // a ≠ 0
                delta = b*b - 4*a*c ;
                if (delta < 0) {
                    System.Console.WriteLine("il n'y a pas de solution dans les reels");
                }
                else { // delta ≥ 0
                    x1 = (-b + Math.Sqrt(delta))/ (2*a) ;
                    x2 = (-b - Math.Sqrt(delta))/ (2*a) ;
                    System.Console.WriteLine("il y deux solutions egales a " + x1 + " et " + x2);
                }
            }
        }
    }
}
```

Algorithme

Calcul des nombres de Armstrong

Objectif : On dénomme nombre de Armstrong un entier naturel qui est égal à la somme des cubes des chiffres qui le composent.

Exemple :

$$153 = 1^3 + 5^3 + 3^3$$

153 = 1 + 125 + 27, est un nombre de Armstrong.

Spécifications de l'algorithme :

On sait qu'il n'existe que 4 nombres de Armstrong, et qu'ils ont tous 3 chiffres (ils sont compris entre 100 et 500).

Si l'on suppose qu'un tel nombre est écrit **ijk** (**i** chiffre des centaines, **j** chiffres des dizaines et **k** chiffres des unités), il suffit simplement d'envisager tous les nombres possibles en faisant varier les chiffres entre 0 et 9 et de tester si le nombre est de Armstrong.

Implantation en C#

Ecrivez le programme C# complet qui fournisse les 4 nombres de Armstrong :

Nombres de Armstrong:

153
370
371
407

Proposition de squelette de classe C# à implanter :

```
class ApplicationArmstrong {  
    static void Main(string[] args) {  
        .....  
    }  
}
```

La méthode **Main** calcule et affiche les nombres de Armstrong.

Squelette plus détaillé de la classe C# à implanter :

```
using System;
namespace CsExosAlgo1
{
    class ApplicationArmstrong {
    static void Main(string[] args) {
        int i, j, k, n, somcube;
        System.Console.WriteLine("Nombres de Armstrong:");
        for(i = 1; i<=9; i++)
            for(j = 0; j<=9; j++)
                for(k = 0; k<=9; k++)
                {
                    .....
                }
            }
        }
    }
```

Classe C# solution

```
using System;
namespace CsExosAlgo1
{
    class ApplicationArmstrong {
    static void Main(string[] args) {
        int i, j, k, n, somcube;
        System.Console.WriteLine("Nombres de Armstrong:");
        for(i = 1; i<=9; i++)
            for(j = 0; j<=9; j++)
                for(k = 0; k<=9; k++)
                {
                    n = 100*i + 10*j + k;
                    somcube = i*i*i + j*j*j + k*k*k;
                    if (somcube == n)
                        System.Console.WriteLine(n);
                }
            }
        }
    }
```

Algorithme

Calcul de nombres parfaits

Objectif : On souhaite écrire un programme C# de calcul des n premiers nombres parfaits. Un nombre est dit parfait s'il est égal à la somme de ses diviseurs, 1 compris.

Exemple : $6 = 1+2+3$, est un nombre parfait.

Spécifications de l'algorithme :

l'algorithme retenu contiendra deux boucles imbriquées. Une boucle de comptage des nombres parfaits qui s'arrêtera lorsque le décompte sera atteint, la boucle interne ayant vocation à calculer tous les diviseurs du nombre examiné d'en faire la somme puis de tester l'égalité entre cette somme et le nombre.

Algorithme Parfait

Entrée: $n \in \mathbb{N}$

Sortie: $\text{nbr} \in \mathbb{N}$

Local: $\text{somdiv}, k, \text{compt} \in \mathbb{N}$

début

lire(n);

$\text{compt} \leftarrow 0$;

$\text{nbr} \leftarrow 2$;

Tantque($\text{compt} < n$) **Faire**

$\text{somdiv} \leftarrow 1$;

Pour $k \leftarrow 2$ **jusqu'à** $\text{nbr}-1$ **Faire**

Si $\text{reste}(\text{nbr} \text{ par } k) = 0$ **Alors** // k divise nbr

$\text{somdiv} \leftarrow \text{somdiv} + k$

Fsi

Fpour ;

Si $\text{somdiv} = \text{nbr}$ **Alors**

 écrire(nbr) ;

$\text{compt} \leftarrow \text{compt}+1$;

Fsi;

$\text{nbr} \leftarrow \text{nbr}+1$

Ftant

FinParfait

Implantation en C#

Ecrivez le programme C# complet qui produise le dialogue suivant à l'écran (les caractères gras représentent ce qui est écrit par le programme, les italiques ce qui est entré au clavier) :

Entrez combien de nombre parfaits : 4

6 est un nombre parfait

28 est un nombre parfait

496 est un nombre parfait

8128 est un nombre parfait

Proposition de squelette de classe C# à implanter :

```
class ApplicationParfaits {  
    static void Main(string[] args) {  
        .....  
    }  
}
```

La méthode **Main** calcule et affiche les nombres parfaits.

Classe C# solution

```
using System;  
namespace CsExosAlgo1  
{  
    class ApplicationParfaits {  
        static void Main (string[] args) {  
            int compt = 0, n, k, somdiv, nbr;  
            System.Console.WriteLine("Entrez combien de nombre parfaits : ");  
            n = Int32.Parse( System.Console.ReadLine() );  
            nbr = 2;  
            while (compt != n)  
            { somdiv = 1;  
              k = 2;  
              while(k <= nbr/2 )  
              {  
                  if (nbr % k == 0) somdiv += k ;  
                  k++;  
              }  
              if (somdiv == nbr)  
              { System.Console.WriteLine(nbr+" est un nombre parfait");  
                compt++;  
              }  
              nbr++;  
            }  
        }  
    }  
}
```

La saisie de l'entier **int** n; s'effectue par transtypage grâce à la méthode Parse de la classe Net Framework [Int32](#) du type **int**.

Algorithme

Calcul du pgcd de 2 entiers (méthode Euclide)

Objectif : On souhaite écrire un programme de calcul du pgcd de deux entiers non nuls, en C# à partir de l'algorithme de la méthode d'Euclide. Voici une spécification de l'algorithme de calcul du PGCD de deux nombres (entiers strictement positifs) **a** et **b**, selon cette méthode :

Spécifications de l'algorithme :

```
Algorithme Pgcd
Entrée: a,b ∈ N* x N*
Sortie: pgcd ∈ N
Local: r,t ∈ N x N

début
lire(a,b);
Si ba Alors
    t ← a ;
    a ← b ;
    b ← t
Fsi;
Répéter
    r ← a mod b ;
    a ← b ;
    b ← r
jusqu'à r = 0;
pgcd ← a;
écrire(pgcd)
FinPgcd
```

Implantation en C#

Ecrivez le programme C# complet qui produise le dialogue suivant à la console (les caractères gras représentent ce qui est écrit par le programme, les italiques ce qui est entré au clavier) :

```
Entrez le premier nombre : 21
Entrez le deuxième nombre : 45
Le PGCD de 21 et 45 est : 3
```

Proposition de squelette de classe C# à implanter :

```
class ApplicationEuclide {  
    static void Main(string[] args) {  
        .....  
    }  
    static int pgcd (int a, int b) {  
        .....  
    }  
}
```

La méthode **pgcd** renvoie le pgcd des deux entiers p et q .

Classe C# solution

```
using System;  
namespace CsExosAlgo1  
{  
    class ApplicationEuclide {  
        static void Main (string[] args) {  
            System.Console.WriteLine("Entrez le premier nombre : ");  
            int p = Int32.Parse( System.Console.ReadLine() );  
            System.Console.WriteLine("Entrez le deuxième nombre : ");  
            int q = Int32.Parse( System.Console.ReadLine() );  
            if (p*q!=0)  
                System.Console.WriteLine("Le pgcd de "+p+" et de "+q+" est "+pgcd(p,q));  
            else  
                System.Console.WriteLine("Le pgcd n'existe pas lorsque l'un des deux nombres est nul !");  
        }  
  
        static int pgcd (int a , int b) {  
            int r,t ;  
            if ( b>a) {  
                t = a;  
                a = b;  
                b = t;  
            }  
            do {  
                r = a % b;  
                a = b;  
                b = r;  
            } while(r !=0);  
            return a ;  
        }  
    }  
}
```

Algorithme

Calcul du pgcd de 2 entiers (méthode Egyptienne)

Objectif : On souhaite écrire un programme de calcul du pgcd de deux entiers non nuls, en C# à partir de l'algorithme de la méthode dite "égyptienne". Voici une spécification de l'algorithme de calcul du PGCD de deux nombres (entiers strictement positifs) p et q , selon cette méthode :

Spécifications de l'algorithme :

```
Lire (p, q) ;  
  
Tantque p ≠ q faire  
  Si p > q alors  
    p ← p - q  
  sinon  
    q ← q - p  
  FinSi  
FinTant;  
Ecrire( " PGCD = ", p )
```

Implantation en C#

Ecrivez le programme C# complet qui produise le dialogue suivant à la console (les caractères gras représentent ce qui est écrit par le programme, les italiques ce qui est entré au clavier) :

```
Entrez le premier nombre : 21  
Entrez le deuxième nombre : 45  
Le PGCD de 21 et 45 est : 3
```

Proposition de squelette de classe C# à implanter :

```
class ApplicationEgyptien {  
  static void Main(String[] args) {  
    .....  
  }  
  static int pgcd (int p, int q) {  
    .....  
  }  
}
```

La méthode **pgcd** renvoie le pgcd des deux entiers p et q .

Classe C# solution

```
using System;
namespace CsExosAlgo1
{
    class ApplicationEgyptien {
        static void Main (string[] args) {
            System.Console.WriteLine("Entrez le premier nombre : ");
            int p = Int32.Parse( System.Console.ReadLine() );
            System.Console.WriteLine("Entrez le deuxième nombre : ");
            int q = Int32.Parse( System.Console.ReadLine() );
            if ( p*q != 0 )
                System.Console.WriteLine("Le pgcd de "+p+" et de "+q+" est "+pgcd(p,q));
            else
                System.Console.WriteLine("Le pgcd n'existe pas lorsque l'un des deux nombres est nul !");
        }

        static int pgcd (int p, int q) {
            while ( p != q )
            {
                if (p > q) p -= q;
                else q -= p;
            }
            return p;
        }
    }
}
```

Algorithme

Calcul de nombres premiers (boucles **while** et **do...while**)

Objectif : On souhaite écrire un programme C# de calcul et d'affichage des n premiers nombres premiers. Un nombre entier est premier s'il n'est divisible que par 1 et par lui-même On opérera une implantation avec des boucles **while** et **do...while**.

Exemple : 31 est un nombre premier

Spécifications de l'algorithme :

Algorithme Premier

Entrée: $n \in \mathbb{N}$

Sortie: $\text{nbr} \in \mathbb{N}$

Local: $\text{Est_premier} \in \{\text{Vrai}, \text{Faux}\}$
 $\text{divis}, \text{compt} \in \mathbb{N}^2$;

début

lire(n);

$\text{compt} \leftarrow 1$;

ecrire(2);

$\text{nbr} \leftarrow 3$;

Tantque($\text{compt} < n$) **Faire**

divis $\leftarrow 3$;

$\text{Est_premier} \leftarrow \text{Vrai}$;

Répéter

Si $\text{reste}(\text{nbr par divis}) = 0$ **Alors**

$\text{Est_premier} \leftarrow \text{Faux}$

Sinon

divis $\leftarrow \text{divis} + 2$

Fsi

jusqu'à ($\text{divis} > \text{nbr} / 2$) **ou** ($\text{Est_premier} = \text{Faux}$);

Si $\text{Est_premier} = \text{Vrai}$ **Alors**

ecrire(nbr);

$\text{compt} \leftarrow \text{compt} + 1$

Fsi;

$\text{nbr} \leftarrow \text{nbr} + 1$

Ftant

FinPremier

Implantation en C#

Ecrivez le programme C# complet qui produise le dialogue suivant à la console (les caractères gras représentent ce qui est écrit par le programme, les italiques ce qui est entré au clavier) :

Combien de nombres premiers : 5

2
3
5
7
11

Classe C# solution

Avec une boucle **while** et une boucle **do...while** imbriquée :

On étudie la primalité de tous les nombres systématiquement

```
using System;
namespace CsExosAlgo1
{
    class ApplicationComptPremiers1 {
        static void Main(string[] args) {
            int divis, nbr, n, compt = 0 ;
            bool Est_premier;
            System.Console.Write("Combien de nombres premiers : ");
            n = Int32.Parse( System.Console.ReadLine() );
            System.Console.WriteLine( 2 );
            nbr = 3;
            while (compt < n-1)
            {
                divis = 2 ;
                Est_premier = true;
                do
                {
                    if (nbr % divis == 0) Est_premier=false;
                    else divis = divis+1 ;
                }
                while ((divis <= nbr/2) && (Est_premier == true));
                if (Est_premier)
                {
                    compt++;
                    System.Console.WriteLine( nbr );
                }
                nbr++ ;
            }
        }
    }
}
```

La méthode **Main** affiche la liste des nombres premiers demandés.

Algorithme

Calcul de nombres premiers (boucles for)

Objectif : On souhaite écrire un programme C# de calcul et d'affichage des n premiers nombres premiers. Un nombre entier est premier s'il n'est divisible que par 1 et par lui-même. On opérera une implantation avec des boucles for imbriquées.

Exemple : 23 est un nombre premier

Spécifications de l'algorithme : (On étudie la primalité des nombres uniquement impairs)

```
Algorithme Premier
  Entrée:  $n \in \mathbb{N}$ 
  Sortie:  $\text{nbr} \in \mathbb{N}$ 
  Local:  $\text{Est\_premier} \in \{\text{Vrai}, \text{Faux}\}$ 
          $\text{divis}, \text{compt} \in \mathbb{N}^2$ ;

  début
    lire( $n$ );
    compt  $\leftarrow$  1;
    écrire(2);
     $\text{nbr} \leftarrow$  3;
    Tantque( $\text{compt} < n$ ) Faire
       $\text{divis} \leftarrow$  3;
       $\text{Est\_premier} \leftarrow$  Vrai;
      Répéter
        Si  $\text{reste}(\text{nbr} \text{ par } \text{divis}) = 0$  Alors
           $\text{Est\_premier} \leftarrow$  Faux
        Sinon
           $\text{divis} \leftarrow \text{divis} + 2$ 
        Fsi
      jusqu'à ( $\text{divis} > \text{nbr} / 2$ ) ou ( $\text{Est\_premier} = \text{Faux}$ );
      Si  $\text{Est\_premier} = \text{Vrai}$  Alors
        écrire( $\text{nbr}$ );
        compt  $\leftarrow$  compt + 1
      Fsi;
       $\text{nbr} \leftarrow \text{nbr} + 2$  // nbr impairs
    Ftant
  FinPremier
```

Implantation en C#

Ecrivez le programme C# complet qui produise le dialogue suivant à la console (les caractères gras représentent ce qui est écrit par le programme, les italiques ce qui est entré au clavier) :

Combien de nombres premiers : 5

2
3
5
7
11

Classe C# solution

Avec deux boucles for imbriquées :

On étudie la primalité des nombres uniquement impairs

```
using System;
namespace CsExosAlgo1
{
    class ApplicationComptPremiers2 {
        static void Main(string[] args) {
            int divis, nbr, n, compt = 0 ;
            bool Est_premier;
            System.Console.WriteLine("Combien de nombres premiers : ");
            n = Int32.Parse( System.Console.ReadLine() );
            System.Console.WriteLine( 2 );
            //-- primalité uniquement des nombres impairs
            for( nbr = 3; compt < n-1; nbr += 2 )
            { Est_premier = true;
              for (divis = 2; divis<= nbr/2; divis++ )
                if ( nbr % divis == 0 )
                { Est_premier = false;
                  break;
                }
              if (Est_premier)
              {
                  compt++;
                  System.Console.WriteLine( nbr );
              }
            }
        }
    }
}
```

La méthode **Main** affiche la liste des nombres premiers demandés.

Le fait de n'étudier la primalité que des nombres impairs accélère la vitesse d'exécution du programme, il est possible d'améliorer encore cette vitesse en ne cherchant que les diviseurs dont le carré est inférieur au nombre (test : **jusqu'à** ($\text{divis}^2 > \text{nbr}$) **ou** ($\text{Est_premier}=\text{Faux}$)).

Algorithme

Calcul du nombre d'or

Objectif : On souhaite écrire un programme C# qui calcule le nombre d'or utilisé par les anciens comme nombre idéal pour la sculpture et l'architecture. Si l'on considère deux suites numériques (U) et (V) telles que pour n strictement supérieur à 2 :

$$\begin{aligned}U_n &= U_{n-1} + U_{n-2} \\ \text{et} \\ V_n &= U_n / U_{n-1}\end{aligned}$$

On montre que la suite (V) tend vers une limite appelée nombre d'or (nbr d'Or = 1,61803398874989484820458683436564).

Spécifications de l'algorithme :

n, U_n, U_{n1}, U_{n2} : sont des entiers naturels

V_n, V_{n1}, ε : sont des nombres réels

lire(ε); // *précision demandée*

$U_{n2} \leftarrow 1;$

$U_{n1} \leftarrow 2;$

$V_{n1} \leftarrow 2;$

$n \leftarrow 2;$ // *rang du terme courant*

Itération

$n \leftarrow n + 1;$

$U_n \leftarrow U_{n1} + U_{n2};$

$V_n \leftarrow U_n / U_{n1};$

si $|V_n - V_{n1}| \leq \varepsilon$ **alors** Arrêt de la boucle ; // *la précision est atteinte*

sinon

$U_{n2} \leftarrow U_{n1};$

$U_{n1} \leftarrow U_n;$

$V_{n1} \leftarrow V_n;$

fsi

fin Itération

ecrire (V_n, n);

Écrire un programme fondé sur la spécification précédente de l'algorithme du calcul du nombre d'or. Ce programme donnera une valeur approchée avec une précision fixée de ε du nombre d'or. Le programme indiquera en outre le rang du dernier terme de la suite correspondant.

Implantation en C#

On entre au clavier un nombre réel ci-dessous 0.00001, pour la précision choisie (ici 5 chiffres après la virgule), puis le programme calcule et affiche le Nombre d'or (les caractères gras représentent ce qui est écrit par le programme, les italiques ce qui est entré au clavier) :

Précision du calcul ? : *0.00001*
Nombre d'Or = **1.6180328** // rang=**14**

Classe C# solution

Avec un boucle for :

```
using System;
namespace CsExosAlgo1
{
    class AppliNombredOr {
        static void Main(string[] args) {
            int n, Un, Un1=2, Un2=1 ;
            double Vn,Vn1=2, Eps ;
            System.Console.Write("Précision du calcul ? : ");
            //-- précision demandée (exemple 1e-4 ou 1e-2 ou 0,0001, ...) :
            Eps = Double.Parse( System.Console.ReadLine() );
            for (n=2; ; n++) //n est le rang du terme courant
            {
                Un = Un1 + Un2;
                Vn =(double)Un / (double)Un1;
                if (Math.Abs(Vn - Vn1) <= Eps) break;
                else
                {
                    Un2 = Un1;
                    Un1 = Un;
                    Vn1 = Vn;
                }
            }
            System.Console.WriteLine("Nombre d'Or = " + Vn+" // rang="+n);
        }
    }
}
```

Remarquons que nous proposons une boucle **for** ne contenant pas de condition de rebouclage dans son en-tête (donc en apparence infinie), puisque nous effectuerons le test "**si** $|V_n - V_{n1}| \leq Eps$ **alors Arrêt de la boucle**" qui permet l'arrêt de la boucle. Dans cette éventualité, la boucle **for** devra donc contenir dans son corps, une instruction de rupture de séquence.

Algorithme

Conjecture de Goldbach

Objectif : On souhaite écrire un programme C# afin de vérifier sur des exemples, la conjecture de Goldbach (1742), soit : "Tout nombre pair est décomposable en la somme de deux nombres premiers".

Dans cet exercice nous réutilisons un algorithme déjà traité (algorithme du test de la primalité d'un nombre entier), nous rappelons ci-après un algorithme indiquant si un entier "nbr" est premier ou non :

```
Algorithme Premier
  Entrée: nbr ∈ N
  Local: Est_premier ∈ {Vrai , Faux}
          divis,compt ∈ N2;

début
  lire(nbr);
  divis ← 3;
  Est_premier ← Vrai;
  Répéter
    Si reste(nbr par divis) = 0 Alors
      Est_premier ← Faux
    Sinon
      divis ← divis+2
    Fsi
  jusqu'à (divis > nbr / 2) ou (Est_premier=Faux);
  Si Est_premier = Vrai Alors
    écrire(nbr est premier)
  Sinon
    écrire(nbr n'est pas premier)
  Fsi
FinPremier
```

Spécifications de l'algorithme de Goldbach :

En deux étapes :

1. On entre un nombre pair n au clavier, puis on génère tous les couples (a,b) tels que $a + b = n$, en faisant varier a de 1 à $n/2$. Si l'on rencontre un couple tel que a et b soient simultanément premiers la conjecture est vérifiée.
2. On peut alors, au choix soit arrêter le programme, soit continuer la recherche sur un autre nombre pair.

Exemple :

Pour $n = 10$, on génère les couples :

(1,9), (2,8), (3,7), (4,6), (5,5)

on constate que la conjecture est vérifiée, et on écrit :

$$10 = 3 + 7$$

$$10 = 5 + 5$$

Conseils :

Il faudra traduire cet algorithme en fonction recevant comme paramètre d'entrée le nombre entier dont on teste la primalité, et renvoyant un booléen **true** ou **false** selon que le nombre entré a été trouvé ou non premier

On écrira la méthode booléenne **EstPremier** pour déterminer si un nombre est premier ou non, et la méthode **generCouples** qui génère les couples répondant à la conjecture.

Classe C# solution

```
using System;
namespace CsExosAlgo1
{
    class ApplicationGoldBach {
        static void Main(string[] args) {
            int n;
            System.Console.WriteLine("Entrez un nombre pair (0 pour finir) :");
            while ( (n = Int32.Parse( System.Console.ReadLine() )) != 0 ){
                generCouples(n); }
        }
        static bool EstPremier(int m) {
            int k ;
            for (k = 2 ; k <= m / 2 ; k++) {
                if (m % k == 0) {
                    return false;
                }
            }
            return true;
        }
        static void generCouples(int n) {
            if (n % 2 == 0) {
                for (int a = 1; a <= n/2; a++) {
                    int b;
                    b = n - a;
                    if ( EstPremier(a) && EstPremier(b) ) {
                        System.Console.WriteLine(n+" = "+a+" + "+b);
                    }
                }
            }
            else System.Console.WriteLine("Votre nombre n'est pas pair !");
        }
    }
}
```

Algorithme

Méthodes d'opérations sur 8 bits

Objectif : On souhaite écrire une application de manipulation interne des bits d'une variable entière non signée sur 8 bits. Le but de l'exercice est de construire une famille de méthodes de travail sur un ou plusieurs bits d'une mémoire. L'application à construire contiendra 9 méthodes :

Spécifications des méthodes :

1. Une méthode **BitSET** permettant de mettre à **1** un bit de rang fixé.
2. Une méthode **BitCLR** permettant de mettre à **0** un bit de rang fixé.
3. Une méthode **BitCHG** permettant de remplacer un bit de rang fixé par son complément.
4. Une méthode **SetValBit** permettant de modifier un bit de rang fixé.
5. Une méthode **DecalageD** permettant de décaler les bits d'un entier, sur la droite de **n** positions (introduction de **n** zéros à gauche).
6. Une méthode **DecalageG** permettant de décaler les bits d'un entier, sur la gauche de **n** positions (introduction de **n** zéros à droite).
7. Une méthode **BitRang** renvoyant le bit de rang fixé d'un entier.
8. Une méthode **ROL** permettant de décaler avec rotation, les bits d'un entier, sur la droite de **n** positions (réintroduction à gauche).
9. Une méthode **ROR** permettant de décaler avec rotation, les bits d'un entier, sur la gauche de **n** positions (réintroduction à droite).

Exemples de résultats attendus :

Prenons une variable X entier (par exemple sur 8 bits) X = **11100010**

1. **BitSET** (X,3) = X = **11101010**
2. **BitCLR** (X,6) = X = **10100010**
3. **BitCHG** (X,3) = X = **11101010**
4. **SetValBit**(X,3,1) = X = **11101010**
5. **DecalageD** (X,3) = X = **00011100**
6. **DecalageG** (X,3) = X = **00010000**
7. **BitRang** (X,3) = 0 ; BitRang (X,6) = 1 ...
8. **ROL** (X,3) = X = **00010111**
9. **ROR** (X,3) = X = **01011100**

Implantation en C#

La conception de ces méthodes ne dépendant pas du nombre de bits de la mémoire à opérer on choisira le type `int` comme base pour une mémoire. Ces méthodes sont classiquement des outils de manipulation de l'information au niveau du bit.

Lors des jeux de tests pour des raisons de simplicité de lecture il est conseillé de ne rentrer que des valeurs entières portant sur 8 bits.

Il est bien de se rappeler que le type primaire **int** est un type entier signé sur 32 bits (représentation en complément à deux).

Proposition de squelette de classe C# à implanter :

```
class Application8Bits {
    static void Main(string [ ] args){
        .....    }
    static int BitSET (int nbr, int num) { .....    }
    static int BitCLR (int nbr, int num) { .....    }
    static int BitCHG (int nbr, int num) { .....    }
    static int SetValBit (int nbr, int rang, int val) { .....    }
    static int DecalageD (int nbr, int n) { .....    }
    static int DecalageG (int nbr, int n) { .....    }
    static int BitRang (int nbr, int rang) { .....    }
    static int ROL (int nbr, int n) { .....    }
    static int ROR (int nbr, int n) { .....    }
}
```

Classe C# solution

```
using System;
namespace CsExosAlgo1
{
    class Application8Bits
    {
        // Int32.MAX_VALUE : 32 bit = 2147483647
        // Int64.MAX_VALUE : 64 bits = 9223372036854775808
```

```
        static void Main(string[] args){
            int n,p,q,r,t;
            n =9;// 000...1001
            System.Console.WriteLine("n=9 : n="+toBinaryString(n));
            p = BitCLR(n,3);// p=1
            System.Console.WriteLine("BitCLR(n,3)="+toBinaryString(p));
            q = BitSET(n,2);// q=13
            System.Console.WriteLine("BitSET(n,2)="+toBinaryString(q));
            r = BitCHG(n,3);// r=1
```

```

System.Console.WriteLine("BitCHG(n,3) =" + toBinaryString(r));
t = BitCHG(n,2); // t=13
System.Console.WriteLine("BitCHG(n,2) =" + toBinaryString(t));
System.Console.WriteLine("p = " + p + ", q = " + q + ", r = " + r + ", t = " + t);
n = -2147483648; // 1000.....00 entier minimal
System.Console.WriteLine("n=-2^31 : n=" + toBinaryString(n));
p=ROL(n,3); // 000...000100 = p=4
System.Console.WriteLine("p = " + p);
n = -2147483648+1; // 1000.....01 entier minimal+1
System.Console.WriteLine("n=-2^31+1 : n=" + toBinaryString(n));
p=ROL(n,3); // 000...0001100 = p=12
System.Console.WriteLine("p = " + p);
n = 3; // 0000.....0 11
System.Console.WriteLine("n=3 : n=" + toBinaryString(n));
p=ROR(n,1); // 100000...001 = p=-2147483647
System.Console.WriteLine("ROR(n,1) = " + p + " = " + toBinaryString(p));
p=ROR(n,2); // 11000...000 = p= -1073741824
System.Console.WriteLine("ROR(n,2) = " + p + " = " + toBinaryString(p));
p=ROR(n,3); // 011000...000 = p= +1610612736 = 2^30+2^29
System.Console.WriteLine("ROR(n,3) = " + p + " = " + toBinaryString(p));
}

```

```

static string toBinaryString ( int n )
{ // renvoie l'écriture de l'entier n en représentation binaire
  string [ ] hexa = { "0000", "0001", "0010", "0011", "0100",
    "0101", "0110", "0111", "1000", "1001", "1010",
    "1011", "1100", "1101", "1110", "1111" };
  string s = string.Format("{0:x}", n), res = "";
  for ( int i = 0; i < s.Length; i++ )
  { char car=(char)s[i];
    if ((car <= '9') && (car >= '0'))
      res = res+hexa[(int)car-(int)'0'];
    if ((car <= 'f') && (car >= 'a'))
      res = res+hexa[(int)car-(int)'a'+10];
  }
  return res;
}

static int BitSET(int nbr, int num)
{ // positionne à 1 le bit de rang num
  int mask;
  mask = 1 << num;
  return nbr | mask;
}

static int BitCLR(int nbr, int num)
{ // positionne à 0 le bit de rang num
  int mask;
  mask = ~ (1 << num);
  return nbr & mask;
}

static int BitCHG(int nbr, int num)
{ // complémente le bit de rang num (0 si bit=1, 1 si bit=0)
  int mask;
  mask = 1 << num;
  return nbr ^ mask;
}

```

```

static int DecalageD (int nbr, int n)
{ // décalage sans le signe de n bits vers la droite
  return nbr >> n ;
}

static int DecalageG (int nbr, int n)
{ // décalage de 2 bits vers la gauche
  return nbr << n ;
}

static int BitRang (int nbr, int rang)
{ //renvoie le bit de rang fixé
  return(nbr >> rang ) %2;
}

static int SetValBit (int nbr, int rang, int val)
{ //positionne à val le bit de rang fixé
  return  val ==0 ? BitCLR( nbr , rang) : BitSET( nbr , rang) ;
}

static int ROL (int nbr, int n)
{ //décalage à gauche avec rotation
  int C;
  int N = nbr;
  for(int i=1; i<=n; i++)
  {
    C = BitRang(N,31);
    N = N <<1;
    N = SetValBit(N,0,C);
  }
  return N ;
}

static int ROR (int nbr, int n)
{ //décalage à droite avec rotation
  int C;
  int N = nbr;
  for(int i=1; i<=n; i++)
  {
    C = BitRang (N,0);
    N = N >> 1;
    N = SetValBit (N,31,C);
  }
  return N ;
}

} //--Application8Bits
}

```

String avec C# phrase palindrome (deux versions)

Une phrase est dite palindrome si en éliminant les blancs entre les mots elle représente la même lecture dans les deux sens :

Exemple : elu par cette crapule ó eluparc ettec rap ule

Voici le squelette du programme C# à écrire :

```
class palindrome
{
    static string compresser ( string s )
    { .....
    }

    static string inverser ( string s )
    { .....
    }
    static void Main ( string [ ] args )
    {
        System.Console.WriteLine ("Entrez une phrase :");
        string phrase = System.Console.ReadLine ();
        string strMot = compresser ( phrase );
        string strInv = inverser ( strMot );
        if( strMot == strInv )
            System.Console.WriteLine ("phrase palindrome !");
        else
            System.Console.WriteLine ("phrase non palindrome !");
        System.Console.ReadLine ();
    }
}
```

Travail à effectuer :

Ecrire les méthode **compresser** et **Inverser** , il est demandé d'écrire **deux versions** de la méthode **Inverser**.

- La première version de la méthode **Inverser** construira une chaîne locale à la méthode caractère par caractère avec une boucle **for** à un seul indice.
- La deuxième version de la méthode **Inverser** modifiera les positions des caractères ayant des positions symétriques dans la chaîne avec une boucle **for** à deux indices et en utilisant un tableau de **char**.

Classe C# solution

Le code de la méthode **compresser** :

```
static string compresser ( string s )
{
    String strLoc = "";
    for( int i = 0 ; i < s.Length ; i ++ )
    {
        if( s[i] != ' ' && s[i] != ' , ' && s[i] != ' \ ' && s[i] != ' . ' )
            strLoc += s[i] ;
    }
    return strLoc ;
}
```

La méthode **compresser** élimine les caractères non recevables comme : blanc, virgule, point et apostrophe de la **String** s passée en paramètre.

Remarquons que l'instruction `strLoc +=s[i]` permet de concaténer les caractères recevables de la chaîne locale `strLoc`, par balayage de la `String s` depuis le caractère de rang 0 jusqu'au caractère de rang `s.Length-1`.

La référence de `String strLoc` pointe à chaque tour de la boucle **for** vers un nouvel objet créé par l'opérateur de concaténation `+`

La première version de la méthode **Inverser** :

```
static string inverser ( string s )
{
    String strLoc = "";
    for( int i = 0 ; i < s.Length ; i ++ )
        strLoc = s[i] + strLoc ;
    return strLoc ;
}
```

La deuxième version de la méthode **Inverser** :

```
static string inverser ( string s )
{
    char [ ] tChar = s.ToCharArray ( );
    char car ;
    for( int i = 0 , j = tChar.Length - 1 ; i < j ; i ++ , j -- )
    {
        car = tChar[i] ;
        tChar[i] = tChar[j] ;
        tChar[j] = car ;
    }
    return new string ( tChar ) ;
}
```

```
for ( int i = 0 , j = tChar.Length - 1 ; i < j ; i ++ , j -- )
```

Trace d'exécution de la boucle sur la chaîne s = "abcdef" :

tChar

a	b	c	d	e	f
f	b	c	d	e	a

i = 0, j=5

a
car

tChar

f	b	c	d	e	a
f	e	c	d	b	a

i = 1, j=4

b
car

tChar

f	e	c	d	b	a
f	e	d	c	b	a

i = 2, j=3

b
car

i = 3, j=2 => i < j est false

String avec C#

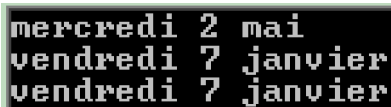
convertir une date numérique en lettres

Objectif : Construire un programme permettant lorsqu'on lui fournit une date sous la forme numérique (3/2/5 où 3 indique le n° du jour de la semaine lundi=1, dimanche=7; le deuxième chiffre 2 indique le jour, enfin le troisième chiffre 5 indique le n° du mois) la convertit en clair (3/2/5 est converti en : mercredi 2 mai).

Proposition de squelette de classe C# à implanter :

```
class Dates
{
    enum LesJours { lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche };
    static string [ ] LesMois = { "janvier", "février", "mars", "avril", "mai", "juin", "juillet",
                                   "août", "septembre", "octobre", "novembre", "décembre" };

    static string mois ( int num ) { .... }
    static string jours ( int num ) { .... }
    static string numjour ( int num ) { .... }
    static void scanDate ( string date, out int jour, out int val, out int mois ) { .... }
    static string ConvertDate ( string date ) { .... }
    static void Main ( string [ ] args )
    {
        System.Console.WriteLine ( ConvertDate ("3/2/05"));
        System.Console.WriteLine ( ConvertDate ("5/7/1"));
        System.Console.WriteLine ( ConvertDate ("05/07/01"));
        System.Console.ReadLine ();
    }
}
```



```
mercredi 2 mai
vendredi 7 janvier
vendredi 7 janvier
```

static string mois (int num)	Si le paramètre est un entier compris entre 1 et 12, elle renvoie le nom du mois (janvier,....,décembre), sinon elle renvoie ###.
static string jours (int num)	Si le paramètre est un entier compris entre 1 et 7, elle renvoie le nom du jour (lundi,....,dimanche), sinon elle renvoie ###.
static string numjour (int num)	Si le paramètre est un entier compris entre 1 et 31, elle le sous forme d'une string, sinon elle renvoie ###.
static void scanDate (string date, out int jour, out int val, out int mois)	Reçoit en entrée une date numérique formatée avec des séparateurs (/,-) et ressort les 3 entiers la composant.
static string ConvertDate (string date)	Reçoit en entrée une date numérique formatée avec des sépaarteurs (/,-) et renvoie sa conversion en forme littérale.

Classe C# solution

```
class Dates
{
    enum LesJours { lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche };
    static string [ ] LesMois = { "janvier", "février", "mars", "avril", "mai", "juin", "juillet",
                                   "août", "septembre", "octobre", "novembre", "décembre" };

    static string mois ( int num )
    {
        if( num >= 0 && num < 12 )
            return LesMois[num];
        else
            return "mois ####";
    }
    static string jours ( int num )
    {
        string [ ] tablej = Enum.GetNames ( typeof ( LesJours ));
        if( num >= 0 && num < 7 )
            return tablej [num];
        else
            return "jour ####";
    }
    static string numjour ( int num )
    {
        if( num >= 1 && num <= 31 )
            return num.ToString ();
        else
            return "n° ####";
    }
    static void scanDate ( string date, out int jour, out int val, out int mois )
    {
        string delimStr = "/.-:";
        char [ ] delimiteurs = delimStr.ToCharArray ();
        string [ ] strInfos = date.Split ( delimiteurs, 3 );
        jour = Convert.ToInt32 ( strInfos[0] ) - 1 ;
        val = Convert.ToInt32 ( strInfos[1] );
        mois = Convert.ToInt32 ( strInfos[2] ) - 1 ;
    }
    static string ConvertDate ( string date )
    {
        int lejour, laval, lemois ;
        scanDate ( date, out lejour, out laval, out lemois );
        return jours ( lejour ) + " " + numjour ( laval ) + " " + mois ( lemois );
    }
    static void Main ( string [ ] args )
    {
        System.Console.WriteLine ( ConvertDate ("3-2-05"));
        System.Console.WriteLine ( ConvertDate ("5/7/1"));
        System.Console.WriteLine ( ConvertDate ("05.07.01"));
        System.Console.ReadLine ();
    }
}
```

Tous les noms du type enum sont automatiquement stockés dans le tableau de string tablej .

La méthode Split de la classe **string** est très utile ici, elle constitue un scanner de chaîne : elle découpe automatiquement la chaîne en 3 éléments dans un tableau de string.

```
mercredi 2 mai
vendredi 7 janvier
vendredi 7 janvier
```

String avec C#

convertir un nombre écrit en chiffres Romains

Objectif : Construire un programme permettant de convertir un nombre romain écrit avec les lettres M,D,C,L,X,V,I en un entier décimal.

Classe C# solutions (récursive et itérative)

```
class Romains
{
    static int tradRom ( char car )
    {
        string carRomainStr = "MDCLXVI";
        if( carRomainStr.IndexOf ( car ) != -1 )
            switch( car )
            {
                case 'M':return 1000 ;
                case 'D':return 500 ;
                case 'C':return 100 ;
                case 'L':return 50 ;
                case 'X':return 10 ;
                case 'V':return 5 ;
                case 'I':return 1 ;
                default : return 0 ;
            }
        else
            return 0 ;
    }
}
```

Méthode récursive	Méthode itérative
<pre>static int recurRom (string chiff, int i) { int valLettre = tradRom (chiff[i]); if(i == chiff.Length - 1) return valLettre; else if (valLettre < tradRom (chiff[i + 1])) return recurRom (chiff, ++ i) - valLettre; else return recurRom (chiff, ++ i) + valLettre; }</pre>	<pre>static int iterRom (string chiff) { int som = 0, valLettre = 0 ; for(int i = 0 ; i < chiff.Length ; i++) { valLettre = tradRom (chiff[i]); if(i == chiff.Length - 1) break; if (valLettre < tradRom (chiff[i + 1])) som = som - valLettre; else som = som + valLettre; } return som + valLettre; }</pre>

```
static void Main ( string [ ] args ) {
    System.Console.WriteLine ( recurRom ("MCDXCIV",0));
    System.Console.WriteLine ( iterRom ("MCDXCIV"));
    System.Console.ReadLine ();
}
}
```

Algorithme

Tri à bulles sur un tableau d'entiers

Objectif : Ecrire un programme C# implémentant l'algorithme du tri à bulles.

Spécifications de l'algorithme :

```
Algorithme Tri_a_Bulles
local: i, j, n, temp ∈ Entiers naturels
Entrée - Sortie : Tab ∈ Tableau d'Entiers naturels de 1 à n éléments
début
pour i de n jusqu'à 1 faire // recommence une sous-suite (a1, a2, ... , ai)
pour j de 2 jusqu'à i faire // échange des couples non classés de la sous-suite
si Tab[ j-1 ] > Tab[ j ] alors // aj-1 et aj non ordonnés
temp ← Tab[ j-1 ];
Tab[ j-1 ] ← Tab[ j ];
Tab[ j ] ← temp //on échange les positions de aj-1 et aj
Fsi
fpour
fpour
Fin Tri_a_Bulles
```

Classe C# solution

```
using System;
namespace CsExosAlgo1
{
class ApplicationTriBulle {
static int[] table; // le tableau à trier, par exemple 19 éléments de l'index 1 à l'index 19
static void AfficherTable () {
// Affichage du tableau
int n = table.Length-1;
for ( int i = 1; i <= n; i++)
System.Console.Write (table[i]+" ", "");
System.Console.WriteLine ();
}

static void InitTable () {
int[] tableau = { 0, 25, 7, 14, 26, 25, 53, 74, 99, 24, 98,
89, 35, 59, 38, 56, 58, 36, 91, 52 };
table = tableau;
}

static void Main(string[] args) {
InitTable ();
System.Console.WriteLine ("Tableau initial :");
AfficherTable ();
TriBulle ();
System.Console.WriteLine ("Tableau une fois trié :");
}
```

```

    AfficherTable ( );
    System.Console.Read();
}
static void TriBulle ( ) {
    // sous-programme de Tri à bulle : on trie les éléments du n°1 au n°19
    int n = table.Length-1;
    for ( int i = n; i >= 1; i--)
        for ( int j = 2; j <= i; j++)
            if (table[j-1] > table[j]){
                int temp = table[j-1];
                table[j-1] = table[j];
                table[j] = temp;
            }
    /* Dans le cas où l'on démarre le tableau à l'indice zéro
       on change les bornes des indices i et j:
       for ( int i = n; i >= 0; i--)
       for ( int j = 1; j <= i; j++)
           if ..... reste identique
    */
}
}
}

```

Tableau initial (n°1 au n°19):

25 , 7 , 14 , 26 , 25 , 53 , 74 , 99 , 24 , 98 , 89 , 35 , 59 , 38 , 56 , 58 , 36 , 91 , 52

Tableau une fois trié (n°1 au n°19) :

7 , 14 , 24 , 25 , 25 , 26 , 35 , 36 , 38 , 52 , 53 , 56 , 58 , 59 , 74 , 89 , 91 , 98 , 99

Autre version depuis l'indice zéro :

Tableau initial (n°0 au n°19):

0, 25 , 7 , 14 , 26 , 25 , 53 , 74 , 99 , 24 , 98 , 89 , 35 , 59 , 38 , 56 , 58 , 36 , 91 , 52

Tableau une fois trié (n°0 au n°19) :

0, 7 , 14 , 24 , 25 , 25 , 26 , 35 , 36 , 38 , 52 , 53 , 56 , 58 , 59 , 74 , 89 , 91 , 98 , 99

Algorithme

Tri par insertion sur un tableau d'entiers

Objectif : Ecrire un programme C# implémentant l'algorithme du tri par insertion.

Spécifications de l'algorithme :

Algorithme Tri_Insertion

local: $i, j, n, v \in$ Entiers naturels

Entrée : Tab $\in$ Tableau d'Entiers naturels de 0 à n éléments

Sortie : Tab $\in$ Tableau d'Entiers naturels de 0 à n éléments (le même tableau)

{ dans la cellule de rang 0 se trouve une sentinelle chargée d'éviter de tester dans la boucle **tantque** .. **faire** si l'indice j n'est pas inférieur à 1, elle aura une valeur inférieure à toute valeur possible de la liste }

début

```
pour  $i$  de 2 jusqu'à  $n$  faire // la partie non encore triée ( $a_i, a_{i+1}, \dots, a_n$ )
     $v \leftarrow \text{Tab}[i]$  ; // l'élément frontière :  $a_i$ 
     $j \leftarrow i$  ; // le rang de l'élément frontière
    Tantque  $\text{Tab}[j-1] > v$  faire // on travaille sur la partie déjà triée ( $a_1, a_2, \dots, a_i$ )
         $\text{Tab}[j] \leftarrow \text{Tab}[j-1]$  ; // on décale l'élément
         $j \leftarrow j-1$  ; // on passe au rang précédent
    FinTant ;
     $\text{Tab}[j] \leftarrow v$  // on recopie  $a_i$  dans la place libérée
fpour
```

Fin Tri_Insertion

Classe C# solution

On utilise une sentinelle placée dans la cellule de rang 0 du tableau, comme le type d'élément du tableau est un **int**, nous prenons comme valeur de la sentinelle une valeur négative très grande par rapport aux valeurs des éléments du tableau; par exemple le plus petit élément du type int, soit la valeur Integer.MIN_VALUE.

```

using System;
namespace CsExosAlgo1
{
class ApplicationTriInsert {
static int[] table; // le tableau à trier, par exemple 19 éléments
/* Tri avec sentinelle :
* dans la cellule de rang 0 se trouve une sentinelle (Int32.MinValue)
* chargée d'éviter de tester dans la boucle tantque .. faire
* si l'indice j n'est pas inférieur à 1, elle aura une valeur
* inférieure à toute valeur possible de la liste.
*/
static void AfficherTable ( ) {
// Affichage du tableau
int n = table.Length-1;
for ( int i = 1; i <= n; i++)
System.Console.Write (table[i]+" , ");
System.Console.WriteLine ( );
}

static void InitTable ( )
{ // sentinelle dans la cellule de rang zéro
int[] tableau = { Int32.MinValue, 25, 7, 14, 26, 25, 53, 74, 99, 24, 98,
89, 35, 59, 38, 56, 58, 36, 91, 52 };
table = tableau;
}

static void Main(string[] args) {
InitTable ( );
System.Console.WriteLine ("Tableau initial :");
AfficherTable ( );
TriInsert ( );
System.Console.WriteLine ("Tableau une fois trié :");
AfficherTable ( );
System.Console.Read();
}

static void TriInsert ( ) {
// sous-programme de Tri par insertion : on trie les éléments du n°1 au n°19
int n = table.Length-1;
for ( int i = 2; i <= n; i++)
{ // la partie non encore triée (ai, ai+1, ... , an)
int v = table[i]; // l'élément frontière : ai
int j = i; // le rang de l'élément frontière
while (table[j-1] > v)
{ //on travaille sur la partie déjà triée (a1, a2, ... , ai)
table[j] = table[j-1]; // on décale l'élément
j = j-1; // on passe au rang précédent
}
table[j] = v; //on recopie ai dans la place libérée
}
}
}
}

```

Tableau initial :

25 , 7 , 14 , 26 , 25 , 53 , 74 , 99 , 24 , 98 , 89 , 35 , 59 , 38 , 56 , 58 , 36 , 91 , 52

Tableau une fois trié :

7 , 14 , 24 , 25 , 25 , 26 , 35 , 36 , 38 , 52 , 53 , 56 , 58 , 59 , 74 , 89 , 91 , 98 , 99

Algorithme

Recherche linéaire dans une table non triée

Objectif : Ecrire un programme C# effectuant une recherche séquentielle dans un tableau linéaire (une dimension) non trié

TABLEAU NON TRIÉ

Spécifications de l'algorithme :

- Soit **t** un tableau d'entiers de **1..n** éléments non rangés.
- On recherche le rang (la place) de l'élément **Elt** dans ce tableau. L'algorithme renvoie le rang (la valeur -1 est renvoyée lorsque l'élément **Elt** n'est pas présent dans le tableau **t**)

Version **Tantque** avec "et alors" (opérateur et optimisé)

```
i ← 1 ;  
Tantque (i ≤ n) et alors (t[i] ≠ Elt) faire  
    i ← i+1  
finTant;  
si i ≤ n alors rang ← i  
sinon rang ← -1  
Fsi
```

Version **Tantque** avec "et" (opérateur et non optimisé)

```
i ← 1 ;  
Tantque (i < n) et (t[i] ≠ Elt) faire  
    i ← i+1  
finTant;  
si t[i] = Elt alors rang ← i  
sinon rang ← -1  
Fsi
```

Version **Tantque** avec sentinelle en fin de tableau (rajout d'une cellule)

```
t[n+1] ← Elt ; // sentinelle rajoutée
```

```

i ← 1 ;
Tantque (i ≤ n) et alors (t[i] ≠ Elt) faire
    i ← i+1
finTant;
si i ≤ n alors rang ← i
sinon rang ← -1
Fsi

```

Version **Pour** avec instruction de sortie (**Sortirsi**)

```

pour i ← 1 jusqu'à n faire
    Sortirsi t[i] = Elt
fpour;
si i ≤ n alors rang ← i
sinon rang ← -1
Fsi

```

Traduire chacune des quatre versions sous forme d'une méthode C#.

Proposition de squelette de classe C# à implanter :

```

class ApplicationRechLin {

    static void AfficherTable ( int[] table ) {
        // Affichage du tableau
    }

    static void InitTable ( ) {
        // remplissage du tableau
    }

    static int RechSeq1( int[] t, int Elt ) {
        // Version Tantque avec "et alors" (opérateur et optimisé)
    }

    static int RechSeq2( int[] t, int Elt ) {
        // Version Tantque avec "et" (opérateur non optimisé)
    }

    static int RechSeq3( int[] t, int Elt ) {
        // Version Tantque avec sentinelle en fin de tableau
    }

    static int RechSeq4( int[] t, int Elt ) {
        // Version Pour avec instruction de sortie break
    }

    static void Main(string[] args)
    {
        InitTable ( );
        System.Console.WriteLine("Tableau initial :");
        AfficherTable (table );
        int x = Int32.Parse(System.Console.ReadLine( )), rang;
        //rang = RechSeq1( table, x );
        //rang = RechSeq2( table, x );
        //rang = RechSeq3( tableSent, x );
        rang = RechSeq4( table, x );
        if (rang > 0)
            System.Console.WriteLine("Élément "+x+" trouvé en : "+rang);
    }
}

```

```

        else System.Console.WriteLine("Élément "+x+" non trouvé !");
        System.Console.Read();
    }
}

```

Classe C# solution

Les différents sous-programmes C# implantant les 4 versions d'algorithme de recherche linéaire (*table non triée*) :

<pre> static int RechSeq1(int[] t, int Elt) { int i = 1; int n = t.Length-1; while ((i <= n) && (t[i] != Elt)) i++; if (i<=n) return i; else return -1; } </pre>	<pre> static int RechSeq2(int[] t, int Elt) { int i = 1; int n = t.Length-1; while ((i < n) & (t[i] != Elt)) i++; if (t[i] == Elt) return i; else return -1; } </pre>
<pre> static int RechSeq3(int[] t, int Elt) { int i = 1; int n = t.Length-2; t[n+1]= Elt ; <i>//sentinelle</i> while ((i <= n) & (t[i] != Elt)) i++; if (i<=n) return i; else return -1; } </pre>	<pre> static int RechSeq4(int[] t, int Elt) { int i = 1; int n = t.Length-1; for(i = 1; i <= n ; i++) if (t[i] == Elt) break; if (i<=n) return i; else return -1; } </pre>

Une classe complète permettant l'exécution des sous-programmes précédents :

```

using System;
namespace CsExosAlgo1
{
    class ApplicationRechLin {
        static int max = 20;
        static int[] table; // cellules à examiner de 1 à 19
        static int[] tableSent = new int[max+1] ; // le tableau à examiner de 1 à 20

        static void AfficherTable (int[] t ) {
            // Affichage du tableau
            int n = t.Length-1;
            for ( int i = 1; i <= n; i++)
                System.Console.Write(t[i]+" , ");
            System.Console.WriteLine( );
        }
        static void InitTable ( ) {
            // remplissage du tableau
            int[] tableau = { 14, 35, 84, 49, 5 , 94, 89, 58, 61, 4 , 39, 58 , 57 , 99 , 12 , 24 , 9 , 81 , 80 };
            table = tableau;
        }
    }
}

```

```

    int n = table.Length-1;
    for ( int i = 1; i <= table.Length-1; i++)
        tableSent[i] = table[i];
}
static int RechSeq1( int[] t, int Elt ) {
    // Version Tantque avec "et alors" (opérateur et optimisé)
    int i = 1; int n = t.Length-1;
    while ((i <= n) && (t[i] != Elt)) i++;
    if (i<=n)
        return i;
    else
        return -1;
}
static int RechSeq2( int[] t, int Elt ) {
    // Version Tantque avec "et" (opérateur non optimisé)
    int i = 1; int n = t.Length-1;
    while ((i < n) & (t[i] != Elt)) i++;
    if (t[i] == Elt)
        return i;
    else
        return -1;
}
static int RechSeq3( int[] t, int Elt ) {
    // Version Tantque avec sentinelle en fin de tableau
    int i = 1; int n = t.Length-2;
    t[n+1]= Elt ; // sentinelle rajoutée
    while ((i <= n) & (t[i] != Elt)) i++;
    if (i<=n)
        return i;
    else
        return -1;
}
static int RechSeq4( int[] t, int Elt ) {
    // Version Pour avec instruction de sortie break
    int i = 1; int n = t.Length-1;
    for(i = 1; i <= n ; i++)
        if (t[i] == Elt)
            break;
    if (i<=n)
        return i;
    else
        return -1;
}
static void Main(String[] args) {
    InitTable ( );
    System.Console.WriteLine("Tableau initial :");
    AfficherTable (table );
    int x = Int32.Parse(System.Console.ReadLine( )), rang;
    //rang = RechSeq1( table, x );
    //rang = RechSeq2( table, x );
    //rang = RechSeq3( tableSent, x );
    rang = RechSeq4( table, x );
    if (rang > 0)
        System.Console.WriteLine("Elément "+x+" trouvé en : "+rang);
    else System.Console.WriteLine("Elément "+x+" non trouvé !");
    System.Console.Read();
}
}
}

```

Algorithme

Recherche linéaire dans table déjà triée

Objectif : Ecrire un programme C# effectuant une recherche séquentielle dans un tableau linéaire (une dimension) trié avant recherche.

Spécifications de l'algorithme :

- Soit **t** un tableau d'entiers de **1..n** éléments rangés par ordre croissant par exemple.
- On recherche le rang (la place) de l'élément **Elt** dans ce tableau. L'algorithme renvoie le rang (la valeur -1 est renvoyée lorsque l'élément **Elt** n'est pas présent dans le tableau **t**)

On peut reprendre sans changement les algorithmes précédents travaillant sur un tableau non trié.

On peut aussi utiliser le fait que le dernier élément du tableau est **le plus grand élément** et s'en servir comme une sorte de **sentinelle**. Ci-dessous deux versions utilisant cette dernière remarque.

Version **Tantque** :

```
si t[n] ≥ Elt alors
  i ← 1 ;
  Tantque t[i] < Elt faire
    i ← i+1;
  finTant;
  si t[i] = Elt alors
    Renvoyer rang ← i // retour du résultat et sortie du programme
  Fsi
Fsi;
Renvoyer rang ← -1 // retour du résultat et sortie du programme
```

Version **pour** :

```
si t[n] ≥ Elt alors rang ← -1
  pour i ← 1 jusqu'à n-1 faire
    Sortirsi t[i] ≥ Elt //sortie de boucle
  fpour;
  si t[i] = Elt alors
    Renvoyer rang ← i // retour du résultat et sortie du programme
  Fsi
Fsi;
Renvoyer rang ← -1 // retour du résultat et sortie du programme
```

Ecrire chacune des méthodes associées à ces algorithmes (prendre soin d'avoir trié le tableau auparavant par exemple par une méthode de tri), squelette de classe proposé :

Classe C# solution

Les deux méthodes C# implantant les 2 versions d'algorithme de recherche linéaire (*table triée*) :

```
static int RechSeqTri1( int[] t, int Elt )
{
    int i = 1; int n = t.Length-1;
    if (t[n] >= Elt) {
        while (t[i] < Elt) i++;
        if (t[i] == Elt)
            return i;
    }
    return -1;
}
```

```
static int RechSeqTri2( int[] t, int Elt )
{
    int i = 1; int n = t.Length-1;
    if (t[n] >= Elt) {
        for (i = 1; i <= n; i++)
            if (t[i] == Elt)
                return i;
    }
    return -1;
}
```

Une classe complète utilisant ces deux méthodes :

```
using System;
namespace CsExosAlgo1
{
    class ApplicationRechLinTrie {

        static int[] table ; //cellules à examiner de 1 à 19

        static void AfficherTable (int[] t) {
            // Affichage du tableau
            int n = t.Length-1;
            for ( int i = 1; i <= n; i++)
                System.Console.Write(t[i]+" , ");
            System.Console.WriteLine( );
        }
        static void InitTable ( ) {
            // remplissage du tableau la cellule de rang 0 est inutilisée
            int[] tableau = { 0, 14, 35, 84, 49, 50, 94, 89, 58, 61, 47,
                             39, 58, 57, 99, 12, 24, 9, 81, 80 };
            table = tableau;
        }
        static void TriInsert ( ) {
            // sous-programme de Tri par insertion :
            int n = table.Length-1;
            for ( int i = 2; i <= n; i++) {
                int v = table[i];
                int j = i;
                while (table[ j-1 ] > v) {
                    table[ j ] = table[ j-1 ];
                    j = j-1;
                }
                table[ j ] = v ;
            }
        }
    }
}
```

```

    }
}

static int RechSeqTri1( int[] t, int Elt ) {
    int i = 1; int n = t.Length-1;
    if (t[n] >= Elt) {
        while (t[i] < Elt) i++;
        if (t[i] == Elt)
            return i;
    }
    return -1;
}

static int RechSeqTri2( int[] t, int Elt ) {
    int i = 1; int n = t.Length-1;
    if (t[n] >= Elt) {
        for (i = 1; i <= n; i++)
            if (t[i] == Elt)
                return i;
    }
    return -1;
}

static void Main(string[] args) {
    InitTable ( );
    System.Console.WriteLine("Tableau initial :");
    AfficherTable (table );
    TriInsert ( );
    System.Console.WriteLine("Tableau trié :");
    AfficherTable (table );
    int x = Int32.Parse(System.Console.ReadLine( )), rang;
    //rang = RechSeqTri1( table, x );
    rang = RechSeqTri2( table, x );
    if (rang > 0)
        System.Console.WriteLine("Élément "+x+" trouvé en : "+rang);
    else System.Console.WriteLine("Élément "+x+" non trouvé !");
    System.Console.Read();
}
}
}

```

Algorithme

Recherche dichotomique dans une table

Objectif : effectuer une recherche dichotomique dans un tableau linéaire déjà trié.

Spécifications de l'algorithme :

- Soit **t** un tableau d'entiers de **1..n** éléments **triés par ordre croissant**.
- On recherche le rang (la place) de l'élément **Elt** dans ce tableau. L'algorithme renvoie le rang (la valeur -1 est renvoyée lorsque l'élément **Elt** n'est pas présent dans le tableau **t**)

Au lieu de rechercher séquentiellement du premier jusqu'au dernier, on compare l'élément **Elt** à chercher au contenu du milieu du tableau. Si c'est le même, on retourne le rang du milieu, sinon l'on recommence sur la première moitié (ou la deuxième) si l'élément recherché est plus petit (ou plus grand) que le contenu du milieu du tableau.

Version itérative

```
bas, milieu, haut, rang : entiers

bas ← 1;
haut ← N;
Rang ← -1;
repéter
    milieu ← (bas + haut) div 2;
    si x = t[milieu] alors
        Rang ← milieu
    sinon si t[milieu] < x alors
        bas ← milieu + 1
    sinon
        haut ← milieu-1
    fsi
fsi
jusqu'à ( x = t[milieu] ) ou ( bas haut )
```

Implanter une méthode en C# que vous nommerez RechDichoIter qui recevant en entrée un tableau et un élément à chercher, renverra le rang de l'élément selon les spécifications ci-haut. N'oubliez pas de trier le tableau avant d'invoquer la méthode de recherche. Ci-dessous une suggestion de rédaction de la méthode **Main** :

```
static void Main(string[] args)
{
    InitTable ();
    System.Console.WriteLine("Tableau initial :");
    AfficherTable ();
}
```

```

TriInsert ( );
System.Console.WriteLine("Tableau trié :");
AfficherTable ( );
int x = Int32.Parse(System.Console.ReadLine( )), rang;
rang = RechDichoIter( table, x );
if (rang < 0)
    System.Console.WriteLine("Élément "+x+" trouvé en : "+rang);
else System.Console.WriteLine("Élément "+x+" non trouvé !");
System.Console.Read();
}

```

Proposition de squelette de classe C# à implanter :

```

class ApplicationRechDicho
{
    static void AfficherTable ( ) {
        // Affichage du tableau ..... }

    static void InitTable ( ) {
        ..... }

    static void TriInsert ( ) {
        // sous-programme de Tri par insertion
        ..... }

    static int RechDichoIter( int[ ] t, int Elt ) {
        // Recherche par dichotomie
        .....
    }

    static void Main(string[ ] args) {
        ..... }
}

```

Classe C# solution

```

using System;
namespace CsExosAlgo1
{
    class ApplicationRechDicho
    {
        static int[ ] table; // le tableau à examiner cellules de 1 à 19
        static void AfficherTable ( )
        {
            // Affichage du tableau
            int n = table.Length-1;
            for ( int i = 1; i <= n; i++)
                System.Console.Write (table[i]+" , ");
            System.Console.WriteLine( );
        }
        static void InitTable ( )
        {
            // La cellule de rang zéro est inutilisée (examen sur 19 éléments)
            int[ ] tableau = { 0 , 53 , 77 , 11 , 72 , 28 , 43 , 65 , 83 , 39 , 73 ,

```

```

            82 , 69 , 65 , 4 , 95 , 46 , 12 , 87 , 75 };
    table = tableau;
}
static void TriInsert ( )
{
    // sous-programme de Tri par insertion :
    int n = table.Length-1;
    for ( int i = 2; i <= n; i++) {
        int v = table[i];
        int j = i;
        while (table[ j-1 ] > v) {
            table[ j ] = table[ j-1 ];
            j = j-1;
        }
        table[ j ] = v ;
    }
}
static int RechDichoIter( int[ ] t, int Elt )
{
    int n = t.Length-1;
    int bas = 1, haut = n, milieu ;
    int Rang = -1;
    do{
        milieu = (bas + haut) / 2;
        if ( Elt == t[milieu]) Rang = milieu ;
        else if ( t[milieu] < Elt ) bas = milieu + 1 ;
        else haut = milieu-1 ;
    }
    while ( ( Elt != t[milieu] ) & ( bas <= haut ) );
    return Rang;
}
static void Main(string[ ] args)
{
    InitTable ( );
    System.Console.WriteLine("Tableau initial :");
    AfficherTable ( );
    TriInsert ( );
    System.Console.WriteLine("Tableau trié :");
    AfficherTable ( );
    int x = Int32.Parse(System.Console.ReadLine( )), rang;
    rang = RechDichoIter( table, x );
    if (rang > 0)
        System.Console.WriteLine("Élément "+x+" trouvé en : "+rang);
    else System.Console.WriteLine("Élément "+x+" non trouvé !");
    System.Console.Read();
}
}

```

Remarque : Ces exercices sont purement académiques et servent à apprendre à utiliser le langage sur des algorithmes classiques, car la classe **Array** contient déjà une méthode static de recherche dichotomique dans un tableau *t* trié au préalable par la méthode static Sort de la même classe **Array** :

```

public static int BinarySearch ( Array t , object elt );

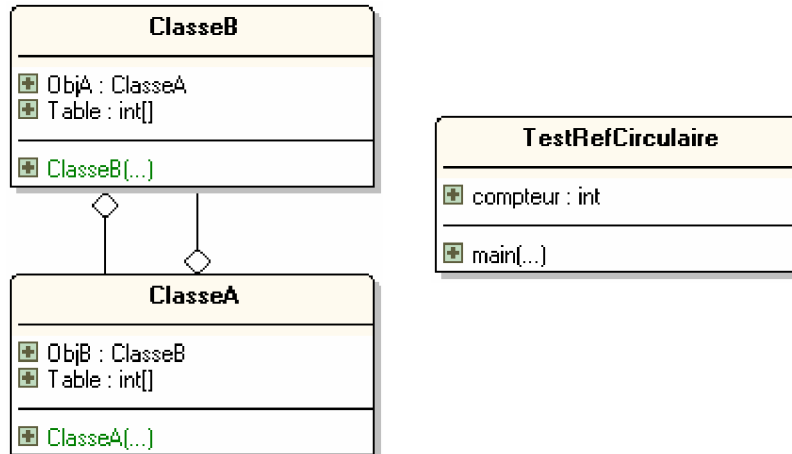
public static void Sort( Array t );

```

Classes, objet et IHM

Problème de la référence circulaire

On donne trois classes ClasseA, ClasseB, TestRefCirculaire :



- La classe ClasseA possède une référence ObjB à un objet public de classe ClasseB, possède un attribut Table qui est un tableau de 50 000 entiers, lorsqu'un objet de ClasseA est construit il incrémente de un le champ static compteur de la classe TestRefCirculaire et instancie la référence ObjB.
- La classe ClasseB possède une référence ObjA à un objet public de classe ClasseA, possède un attribut Table qui est un tableau de 50 000 entiers, lorsqu'un objet de ClasseB est construit il incrémente de un le champ static compteur de la classe TestRefCirculaire et instancie la référence ObjA.
- La classe TestRefCirculaire ne possède qu'un attribut de classe : le champ public entier compteur initialisé à zéro au départ, et la méthode principale Main de lancement de l'application.

Implémentez ces trois classes en ne mettant dans le corps de la méthode Main qu'une seule instruction consistant à instancier un objet local de classe ClasseA, puis exécuter le programme et expliquez les résultats obtenus.

Programme C# solution

Code source demandé :

```
using System;

namespace ConsoleGarbageRefCirc {

    class ClasseA {
```

```

public ClasseB ObjB;
public int[] JTable = new int[50000];

public ClasseA() {
    TestRefCirculaire.compteur++;
    System.Console.WriteLine("Création objet ClasseA n° "+TestRefCirculaire.compteur);
    ObjB = new ClasseB();
}
}

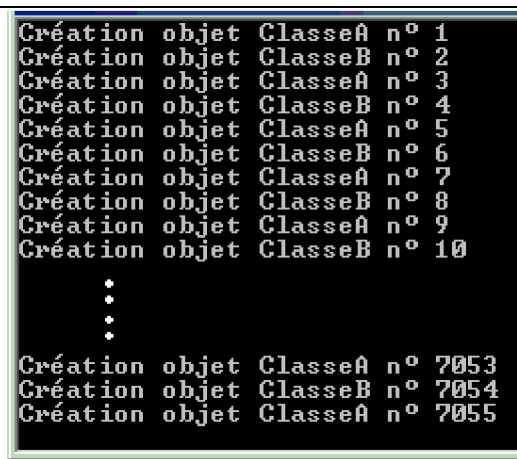
class ClasseB {
    public ClasseA ObjA;
    public int[] JTable = new int[50000];

    public ClasseB() {
        TestRefCirculaire.compteur++;
        System.Console.WriteLine("Création objet ClasseB n° "+TestRefCirculaire.compteur);
        ObjA = new ClasseA();
    }
}

class TestRefCirculaire {
    public static int compteur=0;
    [STAThread]
    static void Main(string[] args) {
        ClasseA ObjA = new ClasseA();
    }
}

```

Résultats d'exécution :



Le système envoie une notification d'exception
OutOfMemoryException, puis arrête le programme :



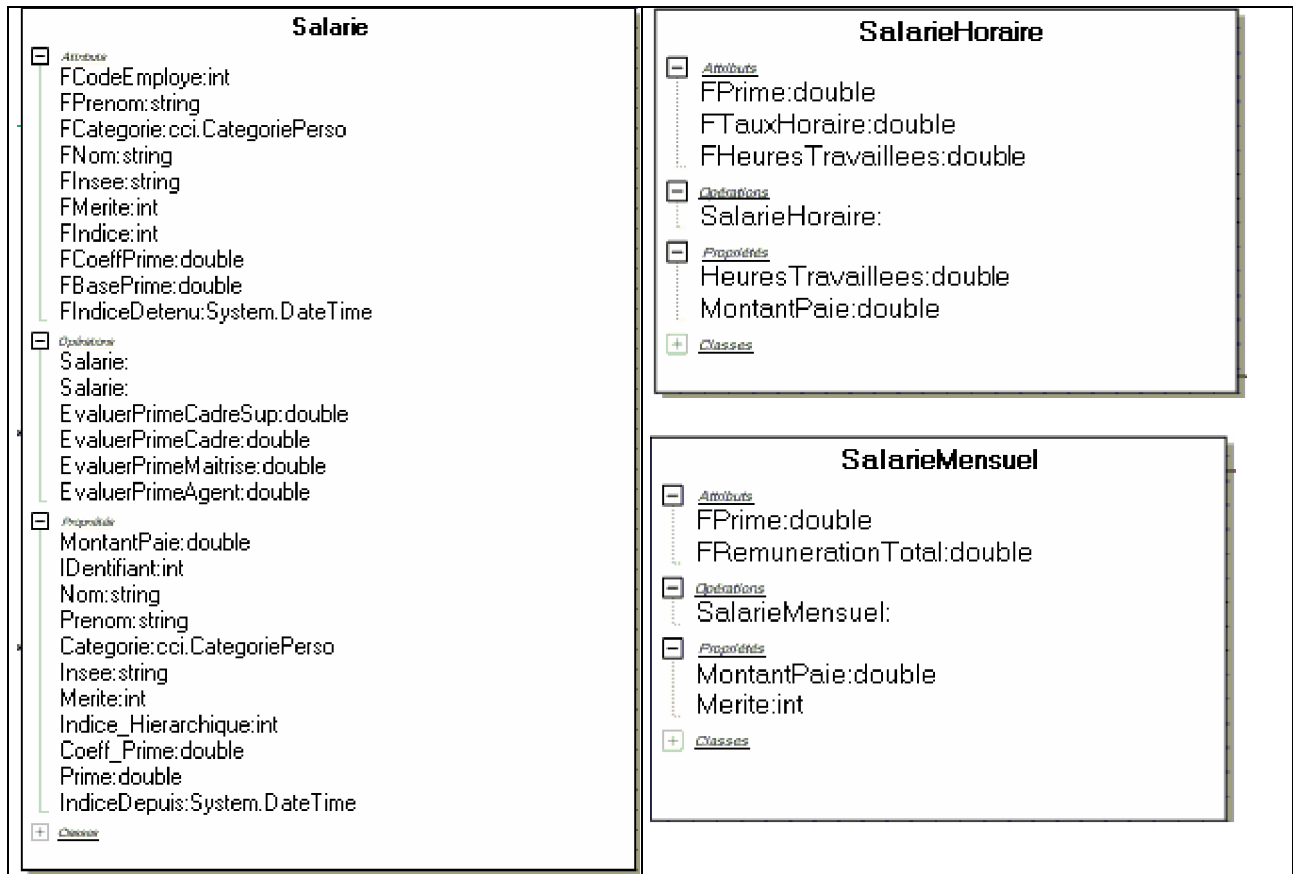
Explications :

L'instanciation d'un objetA provoque l'instanciation d'un objetB qui lui même provoque l'instanciation d'un autre objetA qui à son tour instancie un objetB etc... A chaque instanciation d'un objet de ClasseA ou de ClasseB, un tableau de 50 000 entiers est réservé dans la pile d'exécution, nous voyons sur l'exemple que la 7055^{ème} instanciation a été fatale car la pile a été saturée ! L'instanciation d'un seul objet a provoqué la saturation de la mémoire car les ClasseA et ClasseB sont liées par une association de double référence ou référence circulaire, il faut donc faire attention à ce genre de configuration.

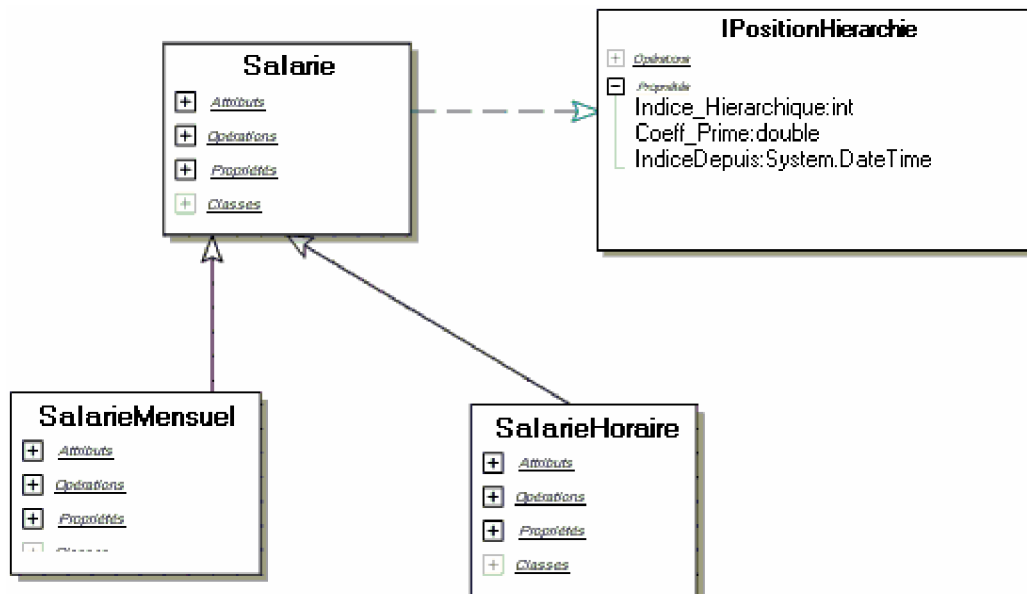
Classes, objet et IHM

Classe de salariés dans une entreprise fictive

Soient les diagrammes de classes suivants censés modéliser le type de salarié employé dans une entreprise. Nous distinguons deux genres de salariés ceux qui sont mensualisés et ceux qui sont payés à l'heure :



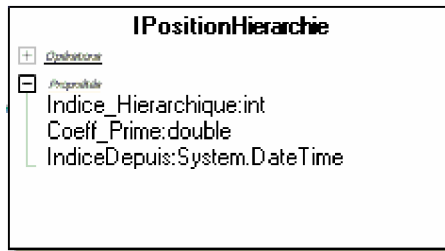
La classe **Salarie** implémente l'interface **IpositionHierarchie** et les classes **SalarieMensuel** et **SalarieHoraire** héritent toutes deux de la classe **Salarie** :



On propose une implémentation partielle des squelettes de ces classes:

```
using System ;
using System.Collections ;
using System.Threading ;
```

```
namespace cci
{
    enum CategoriePerso { Cadre_Sup,Cadre,Maitrise,Agent,Autre }
```



```
/// <summary>
/// Interface définissant les propriétés de position d'un
/// salarié dans la hiérarchie de l'entreprise.
/// </summary>
```

```
interface IPositionHierarchie {
    int Indice_Hierarchique {

}
    double Coeff_Prime {

}
    DateTime IndiceDepui {

}
}
```



```
/// <summary>
/// Classe de base abstraite pour le personnel. Cette classe n'est
/// pas instanciable.
/// </summary>
```

```
abstract class Salarie : IPositionHierarchie
{
    /// attributs identifiant le salarié :
    private int FCodeEmploye ;
    private string FPrenom ;
    private CategoriePerso FCategorie ;
    private string FNom ;
    private string FInsee ;
    protected int FMerite ;
    private int FIndice ;
    private double FCoeffPrime ;
    private double FBasePrime ;
    private DateTime FIndiceDetenu ;

    ///le constructeur de la classe Salarie , payé au mérite :
    public Salarie ( int IDentifiant, string Nom, string Prenom, CategoriePerso Categorie,
    string Insee, int Merite, int Indice, double CoeffPrime ) {

}
    ///le constructeur de la classe Salarie , payé sans mérite :
```

```

public Salarie ( ..... )... {
}
protected double EvaluerPrimeCadreSup ( int coeffMerite ) {
    return ( 100 + coeffMerite * 8 ) * FCoeffPrime * FBasePrime + FIndice * 7 ;
}
protected double EvaluerPrimeCadre ( int coeffMerite ) {
    return ( 100 + coeffMerite * 6 ) * FCoeffPrime * FBasePrime + FIndice * 5 ;
}
protected double EvaluerPrimeMaitrise ( int coeffMerite ) {
    return ( 100 + coeffMerite * 4 ) * FCoeffPrime * FBasePrime + FIndice * 3 ;
}
protected double EvaluerPrimeAgent ( int coeffMerite ) {
    return ( 100 + coeffMerite * 2 ) * FCoeffPrime * FBasePrime + FIndice * 2 ;
}
/// propriété abstraite donnant le montant du salaire
/// (virtual automatiquement)
abstract public double MontantPaie {

}
/// propriété identifiant le salarié dans l'entreprise (lecture) :
public int IDentifiant {

}
/// propriété nom du salarié (lecture /écriture) :
public string Nom {

}
/// propriété prénom du salarié (lecture /écriture) :
public string Prenom {

}
/// propriété catégorie de personnel du salarié (lecture /écriture) :
public CategoriePerso Categorie {

}
/// propriété n° de sécurité sociale du salarié (lecture /écriture) :
public string Insee {

}
/// propriété de point de mérite du salarié (lecture /écriture) ::
public virtual int Merite {

}
/// propriété classement indiciaire dans la hiérarchie (lecture /écriture) :
public int Indice_Hierarchique {
    //--lors de l'écriture : Maj de la date de détention du nouvel indice
}
/// propriété coefficient de la prime en % (lecture /écriture) :
public double Coeff_Prime {

}
/// propriété valeur de la prime selon la catégorie (lecture) :
public double Prime {

}
/// date à laquelle l'indice actuel a été obtenu (lecture /écriture) :
public DateTime IndiceDepuis {

}
}

```



```

/// <summary>
/// Classe du personnel mensualisé. Implémente la propriété abstraite
/// MontantPaie déclarée dans la classe de base (mère).
/// </summary>

```

```

class SalarieMensuel : Salarie
{
    /// attributs du salaire annuel :
    private double FPrime ;
    private double FRemunerationTotal ;

    ///le constructeur de la classe (salarie au mérite) :
    public SalarieMensuel ( int IDentifiant, string Nom, string Prenom,
    CategoriePerso Categorie, string Insee, int Merite, int Indice,
    double CoeffPrime, double RemunerationTotal )... {
        // la prime est calculée
    }

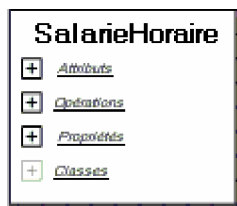
    /// implémentation de la propriété donnant le montant du salaire (lecture) :
    public .... double MontantPaie {

    }

    /// propriété de point de mérite du salarié (lecture /écriture) :
    public ... int Merite {

    }
}

```



```

/// <summary>
/// Classe du personnel horaire. Implémente la propriété abstraite
/// MontantPaie déclarée dans la classe de base (mère).
/// </summary>

```

```

class SalarieHoraire : Salarie
{
    /// attributs permettant le calcul du salaire :
    private double FPrime ;
    private double FTauxHoraire ;
    private double FHeuresTravaillees ;

    ///le constructeur de la classe (salarie non au mérite):
    public SalarieHoraire ( int IDentifiant, string Nom, string Prenom,
    CategoriePerso Categorie, string Insee, double TauxHoraire ) ... {

    }

    /// nombre d'heures effectuées (lecture /écriture) :
    public double HeuresTravaillees {

    }

    /// implémentation de la propriété donnant le montant du salaire (lecture) :
    public override double MontantPaie {

    }
}

```

Implémenter les classes avec le programme de test suivant :

```
class ClassUsesSalarie
{

    static void InfoSalarie ( SalarieMensuel empl )
    {
        Console.WriteLine ("Employé n°" + empl.IDentifiant + ": " + empl.Nom + " / " + empl.Prenom );
        Console.WriteLine (" n° SS : " + empl.Insee );
        Console.WriteLine (" catégorie : " + empl.Categorie );
        Console.WriteLine (" indice hiérarchique : " + empl.Indice_Hierarchique + " , détenu depuis : " +
empl.IndiceDepuis );
        Console.WriteLine (" coeff mérite : " + empl.Merite );
        Console.WriteLine (" coeff prime : " + empl.Coeff_Prime );
        Console.WriteLine (" montant prime annuelle : " + empl.Prime );
        Console.WriteLine (" montant paie mensuelle: " + empl.MontantPaie );

        double coefPrimeLoc = empl.Coeff_Prime ;
        int coefMeriteLoc = empl.Merite ;
        //--impact variation du coef de prime
        for( double i = 0.5 ; i < 1 ; i += 0.1 )
        {
            empl.Coeff_Prime = i ;
            Console.WriteLine (" coeff prime : " + empl.Coeff_Prime );
            Console.WriteLine (" montant prime annuelle : " + empl.Prime );
            Console.WriteLine (" montant paie mensuelle: " + empl.MontantPaie );
        }
        Console.WriteLine (" -----");
        empl.Coeff_Prime = coefPrimeLoc ;
        //--impact variation du coef de mérite
        for( int i = 0 ; i < 10 ; i ++ )
        {
            empl.Merite = i ;
            Console.WriteLine (" coeff mérite : " + empl.Merite );
            Console.WriteLine (" montant prime annuelle : " + empl.Prime );
            Console.WriteLine (" montant paie mensuelle: " + empl.MontantPaie );
        }
        empl.Merite = coefMeriteLoc ;
        Console.WriteLine ("=====");
    }
}

[STAThread]
static void Main ( string [] args )
{
    SalarieMensuel Employe1 = new SalarieMensuel ( 123456, "Euton" , "Jeanne" ,
CategoriePerso.Cadre_Sup, "2780258123456" ,6,700,0.5,50000 );
    SalarieMensuel Employe2 = new SalarieMensuel ( 123457, "Yonaize" , "Mah" ,
CategoriePerso.Cadre, "1821113896452" ,5,520,0.42,30000 );
    SalarieMensuel Employe3 = new SalarieMensuel ( 123457, "Ziaire" , "Marie" ,
CategoriePerso.Maitrise, "2801037853781" ,2,678,0.6,20000 );
    SalarieMensuel Employe4 = new SalarieMensuel ( 123457, "Louga" , "Belle" ,
CategoriePerso.Agent, "2790469483167" ,4,805,0.25,20000 );

    ArrayList ListeSalaries = new ArrayList ();
    ListeSalaries.Add ( Employe1 );
    ListeSalaries.Add ( Employe2 );
    ListeSalaries.Add ( Employe3 );
    ListeSalaries.Add ( Employe4 );
    foreach( SalarieMensuel s in ListeSalaries )
```

```

InfoSalarie ( s);
Console.WriteLine(">>> Promotion indice de " + Employe1.Nom + " dans 2 secondes.");
Thread.Sleep ( 2000 );
Employe1.Indice_Hierarchique = 710 ;
InfoSalarie ( Employe1 );
System.Console.ReadLine ();
}
}

```

Résultats obtenus avec le programme de test précédent :

```

Employé n°123456: Euton / Jeanne
n°SS : 2780258123456
catégorie : Cadre_Sup
indice hiérarchique : 700 , détenu depuis : 15/06/2004 19:24:23
coeff mérite : 6
coeff prime : 0,5
montant prime annuelle : 152900
montant paie mensuelle: 16908,3333333333
coeff prime : 0,5
montant prime annuelle : 152900
montant paie mensuelle: 16908,3333333333
coeff prime : 0,6
montant prime annuelle : 182500
montant paie mensuelle: 19375
coeff prime : 0,7
montant prime annuelle : 212100
montant paie mensuelle: 21841,6666666667
coeff prime : 0,8
montant prime annuelle : 241700
montant paie mensuelle: 24308,3333333333
coeff prime : 0,9
montant prime annuelle : 271300
montant paie mensuelle: 26775
coeff prime : 1
montant prime annuelle : 300900
montant paie mensuelle: 29241,6666666667
-----
coeff mérite : 0
montant prime annuelle : 104900
montant paie mensuelle: 12908,3333333333
coeff mérite : 1
montant prime annuelle : 112900
montant paie mensuelle: 13575
coeff mérite : 2
montant prime annuelle : 120900
montant paie mensuelle: 14241,6666666667
coeff mérite : 3
montant prime annuelle : 128900
montant paie mensuelle: 14908,3333333333
coeff mérite : 4
montant prime annuelle : 136900
montant paie mensuelle: 15575
coeff mérite : 5
montant prime annuelle : 144900
montant paie mensuelle: 16241,6666666667
coeff mérite : 6
montant prime annuelle : 152900
montant paie mensuelle: 16908,3333333333
coeff mérite : 7
montant prime annuelle : 160900
montant paie mensuelle: 17575
coeff mérite : 8
montant prime annuelle : 168900
montant paie mensuelle: 18241,6666666667
coeff mérite : 9

```

montant prime annuelle : 176900
montant paie mensuelle: 18908,3333333333

=====

Employé n°123457: Yonaize / Mah
n°SS : 1821113896452
catégorie : Cadre
indice hiérarchique : 520 , détenu depuis : 15/06/2004 19:24:23
coeff mérite : 5
coeff prime : 0,42
montant prime annuelle : 57200
montant paie mensuelle: 7266,6666666667

... tout le tableau :

coeff mérite : ...
montant prime annuelle ...
montant paie mensuelle: ...

=====

..... tous les autres salariés

=====

>>>> Promotion indice de Euton dans 2 secondes.
Employé n°123456: Euton / Jeanne
n°SS : 2780258123456
catégorie : Cadre_Sup
indice hiérarchique : 710 , détenu depuis : 15/06/2004 19:24:25
coeff mérite : 6
coeff prime : 0,5
montant prime annuelle : 152970
montant paie mensuelle: 16914,1666666667
coeff prime : 0,5
montant prime annuelle : 152970
montant paie mensuelle: 16914,1666666667
coeff prime : 0,6
montant prime annuelle : 182570
montant paie mensuelle: 19380,8333333333
coeff prime : 0,7
montant prime annuelle : 212170
montant paie mensuelle: 21847,5
coeff prime : 0,8
montant prime annuelle : 241770
montant paie mensuelle: 24314,1666666667
coeff prime : 0,9
montant prime annuelle : 271370
montant paie mensuelle: 26780,8333333333
coeff prime : 1
montant prime annuelle : 300970
montant paie mensuelle: 29247,5

coeff mérite : 0
montant prime annuelle : 104970
montant paie mensuelle: 12914,1666666667
coeff mérite : 1
montant prime annuelle : 112970
montant paie mensuelle: 13580,8333333333
coeff mérite : 2
montant prime annuelle : 120970
montant paie mensuelle: 14247,5
coeff mérite : 3
montant prime annuelle : 128970
montant paie mensuelle: 14914,1666666667
coeff mérite : 4
montant prime annuelle : 136970
montant paie mensuelle: 15580,8333333333
coeff mérite : 5
montant prime annuelle : 144970
montant paie mensuelle: 1

Programme C# solution

Les classes et interface de base :

```
using System ;
using System.Collections ;
using System.Threading ;

namespace cci
{
    enum CategoriePerso { Cadre_Sup,Cadre,Maitrise,Agent,Autre }

    /// <summary>
    /// Interface définissant les propriétés de position d'un
    /// salarié dans la hiérarchie de l'entreprise.
    /// </summary>

    interface IPositionHierarchie
    {
        int Indice_Hierarchique {
            get ;
            set ;
        }
        double Coeff_Prime {
            get ;
            set ;
        }
        DateTime IndiceDepuis {
            get ;
            set ;
        }
    }
    /// <summary>
    /// Classe de base abstraite pour le personnel. Cette classe n'est
    /// pas instanciable.
    /// </summary>

    abstract class Salarie : IPositionHierarchie
    {
        /// attributs identifiant le salarié :
        private int FCodeEmploye ;
        private string FPrenom ;
        private CategoriePerso FCategory ;
        private string FNom ;
        private string FInsee ;
        protected int FMerite ;
        private int FIndice ;
        private double FCoeffPrime ;
        private double FBasePrime ;
        private DateTime FIndiceDetenu ;

        ///le constructeur de la classe employé au mérite :
        public Salarie ( int IDentifiant, string Nom, string Prenom, CategoriePerso Categorie,
            string Insee, int Merite, int Indice, double CoeffPrime ) {
            FCodeEmploye = IDentifiant ;
            FNom = Nom ;
            FPrenom = Prenom ;
            FCategory = Categorie ;
            FInsee = Insee ;
        }
    }
}
```

```

FMerite = Merite ;
FIndice = Indice ;
FCoeffPrime = CoeffPrime ;
FIndiceDetenu = DateTime.Now ;
switch ( FCategorie )
{
case CategoriePerso.Cadre_Sup :
    FBasePrime = 2000 ;
    break;
case CategoriePerso.Cadre :
    FBasePrime = 1000 ;
    break;
case CategoriePerso.Maitrise :
    FBasePrime = 500 ;
    break;
case CategoriePerso.Agent :
    FBasePrime = 200 ;
    break;
}
}
///le constructeur de la classe employé sans mérite :
public Salarie ( int IDentifiant, string Nom, string Prenom, CategoriePerso Categorie, string Insee ) :
    this( IDentifiant, Nom, Prenom, Categorie, Insee, 0, 0, 0 ) {
}
protected double EvaluerPrimeCadreSup ( int coeffMerite ) {
    return ( 100 + coeffMerite * 8 ) * FCoeffPrime * FBasePrime + FIndice * 7 ;
}
protected double EvaluerPrimeCadre ( int coeffMerite ) {
    return ( 100 + coeffMerite * 6 ) * FCoeffPrime * FBasePrime + FIndice * 5 ;
}
protected double EvaluerPrimeMaitrise ( int coeffMerite ) {
    return ( 100 + coeffMerite * 4 ) * FCoeffPrime * FBasePrime + FIndice * 3 ;
}
protected double EvaluerPrimeAgent ( int coeffMerite ) {
    return ( 100 + coeffMerite * 2 ) * FCoeffPrime * FBasePrime + FIndice * 2 ;
}
/// propriété abstraite donnant le montant du salaire
/// (virtual automatiquement)
abstract public double MontantPaie {
    get ;
}
/// propriété identifiant le salarié dans l'entreprise :
public int IDentifiant {
    get { return FCodeEmploye ; }
}
/// propriété nom du salarié :
public string Nom {
    get { return FNom ; }
    set { FNom = value ; }
}
/// propriété prénom du salarié :
public string Prenom {
    get { return FPrenom ; }
    set { FPrenom = value ; }
}
/// propriété catégorie de personnel du salarié :
public CategoriePerso Categorie {
    get { return FCategorie ; }
    set { FCategorie = value ; }
}

```

```

/// propriété n° de sécurité sociale du salarié :
public string Insee {
    get { return FInsee ; }
    set { FInsee = value ; }
}

/// propriété de point de mérite du salarié :
public virtual int Merite {
    get { return FMerite ; }
    set { FMerite = value ; }
}

/// propriété classement indiciaire dans la hiérarchie :
public int Indice_Hierarchique {
    get { return FIndice ; }
    set {
        FIndice = value ;
        //--Maj de la date de détention du nouvel indice :
        IndiceDepuis = DateTime.Now ;
    }
}

/// propriété coefficient de la prime en %:
public double Coeff_Prime {
    get { return FCoeffPrime ; }
    set { FCoeffPrime = value ; }
}

/// propriété valeur de la prime :
public double Prime {
    get {
        switch ( FCategorie )
        {
            case CategoriePerso.Cadre_Sup :
                return EvaluerPrimeCadreSup ( FMerite );
            case CategoriePerso.Cadre :
                return EvaluerPrimeCadre ( FMerite );
            case CategoriePerso.Maitrise :
                return EvaluerPrimeMaitrise ( FMerite );
            case CategoriePerso.Agent :
                return EvaluerPrimeAgent ( FMerite );
            default :
                return EvaluerPrimeAgent ( 0 );
        }
    }
}

/// date à laquelle l'indice actuel a été obtenu :
public DateTime IndiceDepuis {
    get { return FIndiceDetenu ; }
    set { FIndiceDetenu = value ; }
}

}

/// <summary>
/// Classe du personnel mensualisé. Implémente la propriété abstraite
/// MontantPaie déclarée dans la classe de base (mère).
/// </summary>

class SalarieMensuel : Salarie
{
    /// attributs du salaire annuel :
    private double FPrime ;
    private double FRemunerationTotal ;

```

```

///le constructeur de la classe (salarié au mérite) :
public SalarieMensuel ( int IDentifiant, string Nom, string Prenom, CategoriePerso Categorie,
    string Insee, int Merite, int Indice, double CoeffPrime, double RemunerationTotal ):
    base ( IDentifiant, Nom, Prenom, Categorie, Insee, Merite, Indice, CoeffPrime ) {
    FPrime = this.Prime ;
    FRemunerationTotal = RemunerationTotal ;
}

/// implémentation de la propriété donnant le montant du salaire :
public override double MontantPaie {
    get { return ( FRemunerationTotal + this.Prime ) / 12 ; }
}

/// propriété de point de mérite du salarié :
public override int Merite {
    get { return FMerite ; }
    set { FMerite = value ; FPrime = this.Prime ; }
}
}

/// <summary>
/// Classe du personnel horaire. Implemente la propriété abstraite
/// MontantPaie déclarée dans la classe de base (mère).
/// </summary>

class SalarieHoraire : Salarie
{
    /// attributs permettant le calcul du salaire :
    private double FPrime ;
    private double FTauxHoraire ;
    private double FHeuresTravaillees ;

    ///le constructeur de la classe (salarié non au mérite):
    public SalarieHoraire ( int IDentifiant, string Nom, string Prenom, CategoriePerso Categorie,
        string Insee, double TauxHoraire ):
        base ( IDentifiant, Nom, Prenom, Categorie, Insee ) {
        FTauxHoraire = TauxHoraire ;
        FHeuresTravaillees = 0 ;
        FPrime = 0 ;
    }

    /// nombre d'heures effectuées :
    public double HeuresTravaillees {
        get { return FHeuresTravaillees ; }
        set { FHeuresTravaillees = value ; }
    }

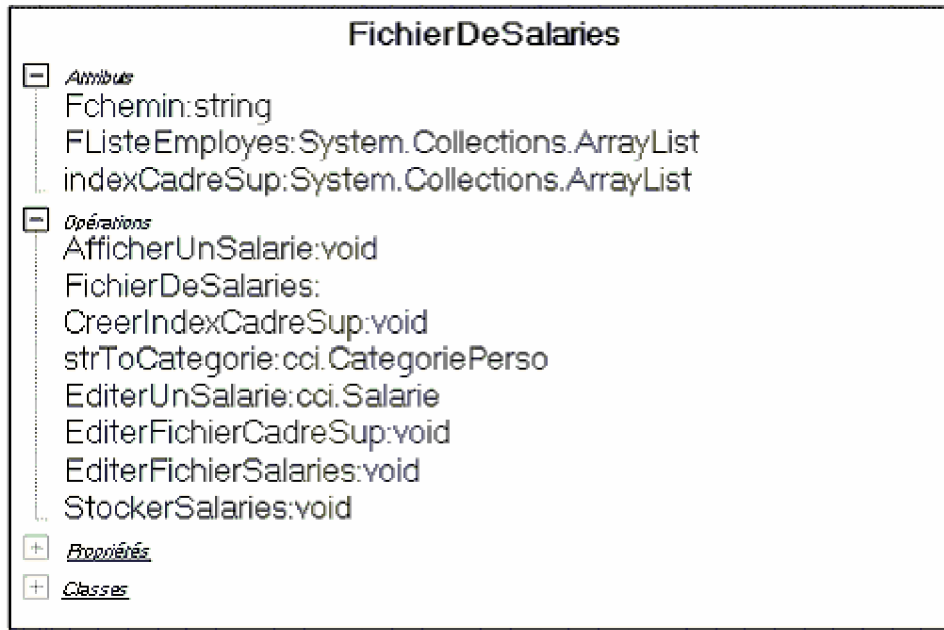
    /// implémentation de la propriété donnant le montant du salaire :
    public override double MontantPaie {
        get { return FHeuresTravaillees * FTauxHoraire + FPrime ; }
    }
}
}

```

Classes, objet et IHM

Classe de salariés dans un fichier de l'entreprise

Nous reprenons les trois classes de l'exercice précédent définissant un salarié horaire et mensualisé dans une entreprise. Nous créons un fichier des salariés de l'entreprise, pour cela nous définissons une classe **FichierDeSalaries** permettant de gérer le fichier des salariés de l'entreprise :



Soit le squelette de la classe **FichierDeSalaries** :

```
using System ;
using System.Collections ;
using System.Threading ;
using System.IO ;

namespace cci
{
    class FichierDeSalaries
    {
        private string Fchemin ;
        private ArrayList FListeEmployes ; // liste des nouveaux employés à entrer dans le fichier
        private ArrayList indexCadreSup ; // Table d'index des cadres supérieurs du fichier

        // méthode static affichant un objet Salarie à la console :
        public static void AfficherUnSalarie ( Salarie Employe ) {
            // pour l'instant un salarié mensualisé seulement
        }
        // constructeur de la classe FichierDeSalaries
        public FichierDeSalaries ( string chemin, ArrayList Liste ) {

        }
        // méthode de création de la table d'index des cadre_sup :
        public void CreerIndexCadreSup ( ) {
```

```

}
// méthode convertissant le champ string catégorie en la constante enum associée
private CatégoriePerso strToCatégorie ( string s ) {

}
// méthode renvoyant un objet SalarieMensuel de rang fixé dans le fichier
private Salarie EditerUnSalarie ( int rang ) {
    SalarieMensuel perso ;

    .....
    perso = new SalarieMensuel ( IDentifiant, Nom, Prenom, Catégorie, Insee,
        Merite, Indice, CoeffPrime, RemunerationTotal );
    .....
    return perso ;
}
// méthode affichant sur la console à partir de la table d'index :
public void EditerFichierCadreSup ()
{
    .....
    foreach( int ind in indexCadreSup )
    {
        AfficherUnSalarie ( EditerUnSalarie ( ind ) );
    }
    .....
}
// méthode affichant sur la console le fichier de tous les salariés :
public void EditerFichierSalaries () {

}
// méthode créant et stockant des salariés dans le fichier :
public void StockerSalaries ( ArrayList ListeEmploy )
{
    .....
    // si le fichier n'existe pas => création du fichier sur disque :
    fichierSortie = File.CreateText ( Fchemin );
    fichierSortie.WriteLine ( "Fichier des personnels");
    fichierSortie.Close ( );
    .....
    // ajout dans le fichier de toute la liste :
    .....
    foreach( Salarie s in ListeEmploy )
    {

    }
    .....
}
}

```

Implémenter la classe FichierDeSalaries avec le programme de test suivant :

```

class ClassUsesSalarie
{
    /// <summary>
    /// Le point d'entrée principal de l'application.
    /// </summary>
    static void InfoSalarie ( SalarieMensuel empl )
    {
        FichierDeSalaries.AfficherUnSalarie ( empl );
        double coefPrimeLoc = empl.Coeff_Prime ;
    }
}

```

```

int coefMeriteLoc = empl.Merite ;
//--impact variation du coef de prime
for( double i = 0.5 ; i < 1 ; i += 0.1 )
{
    empl.Coeff_Prime = i ;
    Console.WriteLine ( "  coeff prime : " + empl.Coeff_Prime );
    Console.WriteLine ( "  montant prime annuelle : " + empl.Prime );
    Console.WriteLine ( "  montant paie mensuelle: " + empl.MontantPaie );
}
Console.WriteLine ( "  -----");
empl.Coeff_Prime = coefPrimeLoc ;
//--impact variation du coef de mérite
for( int i = 0 ; i < 10 ; i ++ )
{
    empl.Merite = i ;
    Console.WriteLine ( "  coeff mérite : " + empl.Merite );
    Console.WriteLine ( "  montant prime annuelle : " + empl.Prime );
    Console.WriteLine ( "  montant paie mensuelle: " + empl.MontantPaie );
}
empl.Merite = coefMeriteLoc ;
Console.WriteLine ( "=====");
}
[STAThread]
static void Main ( string [ ] args )
{
    SalarieMensuel Employe1 = new SalarieMensuel ( 123456, "Euton" , "Jeanne" ,
        CategoriePerso.Cadre_Sup, "2780258123456" ,6,700,0.5,50000 );
    SalarieMensuel Employe2 = new SalarieMensuel ( 123457, "Yonaize" , "Mah" ,
        CategoriePerso.Cadre, "1821113896452" ,5,520,0.42,30000 );
    SalarieMensuel Employe3 = new SalarieMensuel ( 123458, "Ziaire" , "Marie" ,
        CategoriePerso.Maitrise, "2801037853781" ,2,678,0.6,20000 );
    SalarieMensuel Employe4 = new SalarieMensuel ( 123459, "Louga" , "Belle" ,
        CategoriePerso.Agent, "2790469483167" ,4,805,0.25,20000 );

    ArrayList ListeSalaries = new ArrayList ();
    ListeSalaries.Add ( Employe1 );
    ListeSalaries.Add ( Employe2 );
    ListeSalaries.Add ( Employe3 );
    ListeSalaries.Add ( Employe4 );
    foreach( SalarieMensuel s in ListeSalaries )
        InfoSalarie ( s );
    Console.WriteLine ( ">>> Promotion indice de " + Employe1.Nom + " dans 2 secondes.");
    Thread.Sleep ( 2000 );
    Employe1.Indice_Hierarchique = 710 ;
    InfoSalarie ( Employe1 );
    //-----//
    FichierDeSalaries Fiches = new FichierDeSalaries ( "fichierSalaries.txt" ,ListeSalaries );
    Console.WriteLine ( ">>> Attente 3 s pour création de nouveaux salariés");
    Thread.Sleep ( 3000 );
    Employe1 = new SalarieMensuel ( 123460, "Miett" , "Hamas" ,
        CategoriePerso.Cadre_Sup, "1750258123456" ,4,500,0.7,42000 );
    Employe2 = new SalarieMensuel ( 123461, "Kong" , "King" ,
        CategoriePerso.Cadre, "1640517896452" ,4,305,0.62,28000 );
    Employe3 = new SalarieMensuel ( 123462, "Zaume" , "Philippo" ,
        CategoriePerso.Maitrise, "1580237853781" ,2,245,0.8,15000 );
    Employe4 = new SalarieMensuel ( 123463, "Micoton" , "Mylène" ,
        CategoriePerso.Agent, "2850263483167" ,4,105,0.14,12000 );
    ListeSalaries = new ArrayList ();
    ListeSalaries.Add ( Employe1 );

```

```

    ListeSalaries.Add ( Employe2 );
    ListeSalaries.Add ( Employe3 );
    ListeSalaries.Add ( Employe4 );
    Fiches.StockerSalaries ( ListeSalaries );
    Fiches.EditerFichierSalaries ();
    Fiches.CreerIndexCadreSup ();
    Fiches.EditerFichierCadreSup ();
    System.Console.ReadLine ();
}
}

```

Pour tester le programme précédent, on donne le fichier des salariés **fichierSalaries.txt** suivant :

```

Fichier des personnels
123456
Euton
Jeanne
*Cadre_Sup
2780258123456
6
710
15/02/2004 19:52:38
0,5
152970
16914,16666666667
123457
Yonaize
Mah
*Cadre
1821113896452
5
520
15/02/2004 19:52:36
0,42
57200
7266,66666666667
123458
Ziaire
Marie
*Maitrise
2801037853781
2
678
15/02/2004 19:52:36
0,6
34434
4536,16666666667
123459
Louga
Belle
*Agent
2790469483167
4
805
15/02/2004 19:52:36
0,25
7010
2250,83333333333
123460
Miett

```

```

Hamas
*Cadre_Sup
1750258123456
4
500
15/02/2004 19:52:41
0,7
188300
19191,6666666667
123461
Kong
King
*Cadre
1640517896452
4
305
15/02/2004 19:52:41
0,62
78405
8867,08333333333
123462
Zaume
Philippo
*Maitrise
1580237853781
2
245
15/02/2004 19:52:41
0,8
43935
4911,25
123463
Micoton
Mylène
*Agent
2850263483167
4
105
15/02/2004 19:52:41
0,14
3234
1269,5
=====

```

La classe C# *FichierDeSalaries* solution

```

using System ;
using System.Collections ;
using System.Threading ;
using System.IO ;

namespace cci
{
    class FichierDeSalaries
    {
        private string Fchemin ;
        private ArrayList FListeEmployes ;
        private ArrayList indexCadreSup ;
    }
}

```

```

// méthode static affichant un objet Salarie à la console :
public static void AfficherUnSalarie ( Salarie Employe ) {
if( Employe is SalarieMensuel )
{
    SalarieMensuel empl = ( Employe as SalarieMensuel );
    Console.WriteLine ("Employé n°" + empl.IDentifiant + ": " + empl.Nom + " / " + empl.Prenom );
    Console.WriteLine (" n° SS : " + empl.Insee );
    Console.WriteLine (" catégorie : " + empl.Categorie );
    Console.WriteLine (" indice hiérarchique : " + empl.Indice_Hierarchique + " , détenu depuis : "
        + empl.IndiceDepuis );
    Console.WriteLine (" coeff mérite : " + empl.Merite );
    Console.WriteLine (" coeff prime : " + empl.Coeff_Prime );
    Console.WriteLine (" montant prime annuelle : " + empl.Prime );
    Console.WriteLine (" montant paie mensuelle: " + empl.MontantPaie );
}
}

// constructeur de la classeFichierDeSalaries
public FichierDeSalaries ( string chemin, ArrayList Liste ) {
    Fchemin = chemin ;
    FListeEmployes = Liste ;
    StockerSalaries ( FListeEmployes );
}

// méthode de création de la table d'index des cadre_sup :
public void CreerIndexCadreSup () {
    // Ouvre le fichier pour le lire
    StreamReader fichierEntree = File.OpenText ( Fchemin );
    string Ligne ;
    int indexLigne = 0 ;
    indexCadreSup = new ArrayList ( );
    while (( Ligne = fichierEntree.ReadLine () ) != null)
    {
        indexLigne ++ ;
        if("*" + CategoriePerso.Cadre_Sup.ToString () == Ligne )
        {
            Console.WriteLine ("++> " + Ligne + " : " + indexLigne );
            indexCadreSup.Add ( indexLigne - 3 );
        }
    }
    fichierEntree.Close ();
}

// méthode convertissant le champ string catégorie en la constante enum associée
private CategoriePerso strToCategorie ( string s ) {
    switch( s )
    {
        case "*Cadre_Sup":return CategoriePerso.Cadre_Sup ;
        case "*Cadre":return CategoriePerso.Cadre ;
        case "*Maitrise":return CategoriePerso.Maitrise ;
        case "*Agent":return CategoriePerso.Agent ;
        case "*Autre":return CategoriePerso.Autre ;
        default : return CategoriePerso.Autre ;
    }
}

// méthode renvoyant un objet SalarieMensuel de rang fixé dans le fichier
private Salarie EditerUnSalarie ( int rang ) {
    int compt = 0 ;
    string Ligne ;

    int IDentifiant = 0 ;
    string Nom = "" , Prenom = "";

```

```

CategoriePerso Categorie = CategoriePerso.Autre ;
string Insee = "";
int Merite = 0, Indice = 0 ;
DateTime delai = DateTime.Now ;
double CoeffPrime = 0, RemunerationTotal = 0, MontantPaie = 0 ;
SalarieMensuel perso ;
StreamReader f = File.OpenText ( Fchemin );
//System .IFormatProvider format = new System .Globalization.CultureInfo ("fr-FR" , true );

while (( Ligne = f.ReadLine () ) != null)
{
    compt ++ ;
    if ( compt == rang )
    {
        IDentifiant = Convert.ToInt32 ( Ligne );
        Nom = f.ReadLine ();
        Prenom = f.ReadLine ();
        Categorie = strToCategorie ( f.ReadLine () );
        Insee = f.ReadLine ();
        Merite = Convert.ToInt32 ( f.ReadLine () );
        Indice = Convert.ToInt32 ( f.ReadLine () );
        delai = DateTime.Parse ( f.ReadLine () );
        CoeffPrime = Convert.ToDouble ( f.ReadLine () );
        RemunerationTotal = Convert.ToDouble ( f.ReadLine () );
        MontantPaie = Convert.ToDouble ( f.ReadLine () );
        break;
    }
}
f.Close ();
perso = new SalarieMensuel ( IDentifiant, Nom, Prenom, Categorie, Insee, Merite,
    Indice, CoeffPrime, RemunerationTotal );
perso.IndiceDepuis = delai ;
return perso ;
}
// méthode affichant sur la console à partir de la table d'index :
public void EditerFichierCadreSup () {
    StreamReader fichierEntree = File.OpenText ( Fchemin );
    if( indexCadreSup == null)
        CreerIndexCadreSup ();

    foreach( int ind in indexCadreSup )
    {
        AfficherUnSalarie ( EditerUnSalarie ( ind ) );
    }
    fichierEntree.Close ();
}
// méthode affichant sur la console le fichier de tous les salariés :
public void EditerFichierSalaries () {
    // Ouvre le fichier pour le lire
    StreamReader fichierEntree = File.OpenText ( Fchemin );
    string Ligne ;
    while (( Ligne = fichierEntree.ReadLine () ) != null)
    {
        Console.WriteLine ( Ligne );
    }
    fichierEntree.Close ();
}
// méthode créant et stockant des salariés dans le fichier :
public void StockerSalaries ( ArrayList ListeEmploy ) {
    StreamWriter fichierSortie ;

```

```

if ( ! File.Exists ( Fchemin ))
{
    // création du fichier sur disque :
    fichierSortie = File.CreateText ( Fchemin );
    fichierSortie.WriteLine ("Fichier des personnels");
    fichierSortie.Close ();
}

// ajout dans le fichier de tout le :
fichierSortie = File.AppendText ( Fchemin );
if ( FListeEmployes.Count != 0 )
foreach( Salarie s in ListeEmploy )
{
    fichierSortie.WriteLine ( s.IDentifiant );
    fichierSortie.WriteLine ( s.Nom );
    fichierSortie.WriteLine ( s.Prenom );
    fichierSortie.WriteLine ( '*' + s.Categorie.ToString ( ));
    fichierSortie.WriteLine ( s.Insee );
    if( s is SalarieMensuel )
    {
        SalarieMensuel sLoc = ( s as SalarieMensuel );
        fichierSortie.WriteLine ( sLoc.Merite );
        fichierSortie.WriteLine ( sLoc.Indice_Hierarchique );
        fichierSortie.WriteLine ( sLoc.IndiceDepuis );
        fichierSortie.WriteLine ( sLoc.Coeff_Prime );
        fichierSortie.WriteLine ( sLoc.Prime );
    }
    else
        fichierSortie.WriteLine (( s as SalarieHoraire ).HeuresTravaillees );
    fichierSortie.WriteLine ( s.MontantPaie );
}
fichierSortie.Close ();
}
}

```

Classes, objet et IHM

Construction d'un ensemble de caractères

Soit à construire une classe **setOfChar** ensemble de caractères possédant certaines caractéristiques de base d'un ensemble, il est demandé que les opérations suivantes soient présentes : ajouter, enlever un élément de l'ensemble, test d'appartenance d'un élément à l'ensemble, cardinal de l'ensemble (la redondance est acceptée)

Il est aussi demandé à ce que l'ensemble propose deux événements OnInsérer qui se produit lorsque l'on ajoute un nouvel élément à l'ensemble et OnEnlever qui a lieu lorsque l'on supprime un élément de l'ensemble.

La classe **setOfChar** héritera de la classe **CollectionBase** et implémentera une interface événementielle **IEventEnsemble** : `class setOfChar : CollectionBase, IeventEnsemble`.

CollectionBase est une classe de .Net Framework et fournit la classe de base abstract pour une collection fortement typée :

```
System.Object
|__System.Collections.CollectionBase
public abstract class CollectionBase : IList, ICollection, IEnumerable
```

L'interface **IList** représente une liste d'objets accessibles séparément par indexeur et des méthodes classiques de gestion de liste dont nous extrayons ci-dessous les principales utiles, à l'exercice à traiter:

Méthodes publiques

int Add(object valeur);	Ajoute l'élément valeur dans la liste
void Remove(object valeur);	Enlève l'élément valeur dans la liste
bool Contains(object valeur);	La liste contient l'élément valeur .

IeventEnsemble est une interface qui est donnée pour décrire les deux événements auxquels un ensemble doit être sensible :

```
interface IEventEnsemble {
    event EventHandler OnInsérer;
    event EventHandler OnEnlever;
}
```

Question :

compléter dans le squelette de programme ci-après, la classe ensemble de caractère **setOfChar**.

```
using System;
using System.Collections;
```

```
namespace cci
{
```

```
interface IEventEnsemble {
    event EventHandler OnInserer;
    event EventHandler OnEnlever;
}
```

```
public class setOfChar : CollectionBase, IEventEnsemble {

    public event EventHandler OnInserer;
    public event EventHandler OnEnlever;

    public static setOfChar operator + ( setOfChar e1, setOfChar e2) {
        // surcharge de l'opérateur d'addition étendu aux ensembles de char
        ....
    }

    //-- les constructeurs de la classe servent à initialiser l'ensemble :
    public setOfChar() {
        ....
    }

    public setOfChar(string s) : this(s.ToCharArray()) {
        ....
    }

    public setOfChar(char[] t) {
        ....
    }

    public char[] ToArray() {
        // renvoie les éléments de l'ensemble sous forme d'un tableau de char
        ....
    }

    public override string ToString() {
        // renvoie les éléments de l'ensemble sous forme d'une chaîne
        ....
    }

    public int Card {
        // cardinal de l'ensemble
        ....
    }

    protected virtual void Inserer( object sender, EventArgs e ) {
        // lance l'événement OnInserer
        ....
    }

    protected virtual void Enlever( object sender, EventArgs e ) {
        // lance l'événement OnEnlever
        ....
    }

    public char this[ int index ] {
        // indexeur de l'ensemble
        ....
    }
}
```

```

}

public int Add( char value ) {
    // ajoute un élément à l'ensemble
    ....
}

public void Remove( char value ) {
    // enlève un élément à l'ensemble s'il est présent
    ....
}

public bool Contains( char value ) {
    // true si value est dans type setOfChar, false sinon.
    ....
}

protected override void OnInsert( int index, Object value ) {
    ....
}

protected override void OnRemove( int index, Object value ) {
    ....
}
}

```

La méthode **protected virtual void OnInsert** est présente dans la classe **CollectionBase**, elle sert à exécuter des actions avant l'insertion d'un nouvel élément dans la collection. Elle va servir de base au lancement de l'événement **OnInsérer**.

La méthode **protected virtual void OnRemove** est présente dans la classe **CollectionBase**, elle sert à exécuter des actions lors de la suppression d'un élément dans la collection. Elle va servir de base au lancement de l'événement **OnEnlever**.

Classe permettant de tester le fonctionnement de setOfChar

```

public class UtiliseSetOfChar
{

    private static void EnsCarOnEnlever( object sender, EventArgs e ) {
        Console.WriteLine();
        Console.WriteLine("Événement - on va enlever un élément de : "+sender.ToString());
        Console.WriteLine();
    }

    public static void Main() {

        // initialisation
        setOfChar EnsCar1 = new setOfChar();
        setOfChar EnsCar,EnsCar2;

        // ajout d'éléments
        EnsCar1.Add('a');
        EnsCar1.Add('b');
        EnsCar1.Add('c');
        EnsCar1.Add('d');
        EnsCar1.Add('e');
        Console.WriteLine("card="+EnsCar1.Card+" ; "+EnsCar1.ToString());
        EnsCar2=new setOfChar("xyztu#");
        Console.WriteLine("card="+EnsCar2.Card+" ; "+EnsCar2.ToString());
        EnsCar=EnsCar1+EnsCar2;
        EnsCar.OnEnlever += new EventHandler(EnsCarOnEnlever);
        Console.WriteLine("card="+EnsCar.Card+" ; "+EnsCar.ToString());
        Console.WriteLine();

        // Contenu de l'ensemble avec for
        Console.WriteLine( "Contenu de l'ensemble:" );
    }
}

```

```

for(int i=0; i<EnsCar.Card; i++)
    Console.Write( EnsCar[i]+"," );
Console.WriteLine();
// on enlève un élément dans l'ensemble
EnsCar.Remove( 'd' );

// Contenu de l'ensemble avec foreach
Console.WriteLine( "Contenu de l'ensemble après enlèvement de l'élément d : " );
foreach(char elt in EnsCar)
    Console.Write(elt+",");
Console.ReadLine();
}
}
}

```

Résultats de l'exécution :

```

card=5 ; abcde
card=6 ; xyztu#
card=11 ; abcdexyztu#

Contenu de l'ensemble:
a,b,c,d,e,x,y,z,t,u,#.

Événement - on va enlever un élément de : abcdexyztu#

Contenu de l'ensemble après enlèvement de l'élément d :
a,b,c,e,x,y,z,t,u,#.

```

La classe C# *setOfChar* solution

```

public class setOfChar : CollectionBase, IEventEnsemble
{

```

```

    public event EventHandler OnInserer;
    public event EventHandler OnEnlever;

```

```

    public static setOfChar operator + ( setOfChar e1, setOfChar e2) {
        return new setOfChar(e1.ToString()+e2.ToString());
    }

```

```

    public setOfChar()
    { }

```

```

    public setOfChar(string s) : this(s.ToCharArray())
    { }

```

```

    public setOfChar(char[] t) {
        foreach(char car in t)
            this.Add(car);
    }

```

```

    public char[] ToArray()
    {
        char[] t = new char[this.Count];
        for(int i=0; i<this.Count; i++)
            t[i]=this[i];
        return t;
    }

```

On convertit les deux ensembles sous forme de chaînes, on les concatène, puis on appelle le deuxième constructeur. On peut maintenant écrire :
 $E = F + G$ où E, F, G sont des **setOfChar**.

Ce constructeur permet de charger l'ensemble par une chaîne s en appelant le constructeur du dessous par conversion de la chaîne s en tableau de char (méthode ToCharArray).

Le seul constructeur qui a un corps et permettant de charger l'ensemble avec un tableau de caractères.

```

public override string ToString() {
    return new string(this.ToArray());
}

public int Card {
    get {
        return this.Count;
    }
}

protected virtual void Inserir( object sender, EventArgs e ) {
    if ( OnInserir != null )
        OnInserir( sender , e );
}

protected virtual void Enlever( object sender, EventArgs e ) {
    if ( OnEnlever != null )
        OnEnlever( sender , e );
}

public char this[ int index ] {
    get {
        return( (char) List[index] );
    }
    set {
        List[index] = value;
    }
}

public int Add( char value ) {
    return ( List.Add( value ) );
}

public void Remove( char value ) {
    if (Contains(value))
        List.Remove( value );
}

public bool Contains( char value ) {
    return ( List.Contains( value ) );
}

protected override void OnInsert ( int index, Object value ) {
    Inserir( this , EventArgs.Empty );
}

protected override void OnRemove( int index, Object value ) {
    Enlever ( this , EventArgs.Empty );
}
}

```

OnInsert appelle la méthode Inserir qui lance l'éventuel gestionnaire d'événement **OnInserir**.

OnRemove appelle la méthode Enlever qui lance l'éventuel gestionnaire d'événement **OnEnlever**.

Si l'on veut éliminer la redondance d'élément (un élément n'est présent qu'une seule fois dans un ensemble) il faut agir sur la méthode d'ajout (add) et vérifier que l'ajout est possible.

L'ajout est possible si l'élément à ajouter n'est pas déjà contenu dans la liste :

Comme la méthode `add` renvoie le rang d'insertion de l'élément, nous lui faisons renvoyer la valeur -1 lorsque l'élément n'est pas ajouté parce qu'il est déjà présent dans la liste :

```
public int Add( char value ) {  
    if (!Contains(value))  
        return ( List.Add ( value ) );  
    else  
        return -1;  
}
```

Nous pouvons aussi prévoir d'envoyer un message à l'utilisateur de la classe sous forme d'une fenêtre de dialogue l'avertissant du fait qu'un élément était déjà présent et qu'il n'a pas été rajouté. Nous utilisons la classe **MessageBox** de C# qui sert à afficher un message pouvant contenir du texte, des boutons et des symboles :

System.Object
└─ **System.Windows.Forms.MessageBox**

Plus particulièrement, nous utilisons une des surcharges de la méthode **Show** :

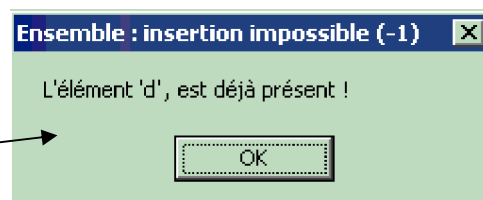
```
public static DialogResult Show( string TexteMess , string caption );
```

Code de la méthode Add (avec message)

```
public int Add( char value ) {  
    if (!Contains(value))  
        return ( List.Add ( value ) );  
    else {  
        MessageBox.Show ("L'élément '" +value.ToString() +"' , est déjà présent !",  
                        "Ensemble : insertion impossible (-1)");  
        return -1;  
    }  
}
```

Voici dans cette éventualité, ce que provoque l'exécution de la sixième ligne du code ci-dessous sur la deuxième demande d'insertion de l'élément 'd' dans l'ensemble `EnsCar1` :

```
EnsCar1.Add('a');  
EnsCar1.Add('b');  
EnsCar1.Add('c');  
EnsCar1.Add('d');  
EnsCar1.Add('e');  
EnsCar1.Add('d');
```



Classes, objet et IHM

Construction d'une classe d'ensemble générique

Soit à construire une classe **setOfObject** d'ensemble plus générale que celle de l'exercice précédent. Nous souhaitons en, effet disposer d'une classe de type ensemble possédant les fonctionnalités de la classe **setOfChar** (ajouter, enlever un élément de l'ensemble, test d'appartenance d'un élément à l'ensemble, cardinal de l'ensemble, non redondance d'un élément), qui puisse accueillir des éléments de même type mais que ce type puisse être n'importe quel type héritant de la classe object.

Cette classe d'ensemble, proposera deux événements OnInsérer qui se produit lorsque l'on ajoute un nouvel élément à l'ensemble et OnEnlever qui a lieu lorsque l'on supprime un élément de l'ensemble.

La classe **setOfObject** héritera de la classe **CollectionBase** et implémentera une interface événementielle **IEventEnsemble** : **class** setOfObject : CollectionBase, IEventEnsemble

Conseils :

Par rapport à l'exercice précédent, il faut faire attention à l'ajout d'un élément du même type que tous ceux qui sont déjà présents dans l'ensemble et refuser un nouvel élément qui n'est pas strictement du même type. Il faut donc utiliser le mécanisme de réflexion de C# (connaître le type dynamique d'un objet lors de l'exécution) contenu dans la classe Type.

Nous proposons un squelette détaillé de la classe **setOfObject** à compléter, les méthodes qui sont identiques à celle de l'exercice précédent ont été mise avec leur code, les autres sont à définir par le lecteur.

Nous souhaitons disposer de plusieurs surcharges du constructeur d'ensemble générique :

constructeur	fonctionnalité
public setOfObject()	Construit un ensemble vide (de n'importe quel type)
public setOfObject (object[] t)	Rempli un ensemble à partir d'un tableau d'object. (le type des éléments de l'ensemble construit est automatiquement celui du premier élément du object[])
public setOfObject (Array t)	Rempli un ensemble à partir d'un tableau de type Array. (le type des éléments de l'ensemble construit est automatiquement celui du premier élément du Array)
public setOfObject (ArrayList t)	Rempli un ensemble à partir d'un tableau de type ArrayList. (le type des éléments de l'ensemble construit est automatiquement celui du premier élément du ArrayList)
public setOfObject (string s)	Rempli un ensemble à partir chaîne. (le type des éléments de l'ensemble construit est automatiquement celui du premier élément, ici char)

Nous proposons enfin, de prévoir un champ protégé nommé FtypeElement qui contient le type de l'élément de l'ensemble qui peut varier au cours du temps Car lorsque l'ensemble est vide il n'a pas de type d'élément c'est l'ajout du premier élément qui détermine le type de l'ensemble et donc des futurs autres éléments à introduire. Ce champ protégé devra être accessible en lecture seulement par tout utilisateur de la classe.

Code à compléter :

```
public class setOfObject : CollectionBase, IEventEnsemble {

    protected Type FTypeElement;

    public Type TypeElement {
        get {
            return FTypeElement;
        }
    }

    public event EventHandler OnInserer;
    public event EventHandler OnEnlever;

    public static setOfObject operator + ( setOfObject e1, setOfObject e2) {
        .....
    }

    public setOfObject() {
    }

    public setOfObject(object[] t) {
        .....
    }

    public setOfObject(Array t) {
        .....
    }

    public setOfObject (ArrayList t) ..... {
    }

    public setOfObject(string s) {
        .....
    }

    public virtual object[] ToArray() {
        .....
    }

    public override string ToString() {
        .....
    }

    public int Card {
        get {
            return this.Count;
        }
    }

    protected virtual void Inserer( object sender, EventArgs e ) {
        if ( OnInserer != null )
            OnInserer( sender , e );
    }

    protected virtual void Enlever( object sender, EventArgs e ) {
        if ( OnEnlever != null )
            OnEnlever( sender , e );
    }
}
```

```

public object this[ int index ] {
    get {
        return( List[index] );
    }
    set {
        List[index] = value;
    }
}

public int Add( object value ) {
    /* ajoute un élément à l'ensemble sinon renvoie -1 si déjà présent
    ou bien renvoie -2 si l'élément n'est pas du même type que les autres */
    ....
}

public void Remove( object value ) {
    // enlève un élément à l'ensemble s'il est présent et de même type
    ....
}

public bool Contains( object value ) {
    // true si value est dans type setOfChar, false sinon.
    return( List.Contains( value ) );
}

protected override void OnInsert( int index, Object value ) {
    Insérer( this , EventArgs.Empty );
}

protected override void OnRemove( int index, Object value ) {
    Enlever( this , EventArgs.Empty );
}
}

```

La classe C# *setOfObject* solution

```

public class setOfObject : CollectionBase, IEventEnsemble {

    protected Type FTypeElement;

    public Type TypeElement {
        get {
            return FTypeElement;
        }
    }

    public event EventHandler OnInsérer;
    public event EventHandler OnEnlever;

    public static setOfObject operator + ( setOfObject e1, setOfObject e2 ) {
        ArrayList t = new ArrayList(e1);
        foreach(object elt in e2)
            t.Add(elt);
        return new setOfObject(t);
    }
}

```

Champ FtypeElement contenant le type de l'ensemble.

Propriété permettant de lire le champ FtypeElement contenant le type de l'ensemble.

On se sert du constructeur fondé sur le remplissage à partir d'un ArrayList.

```
public setOfObject ( ) {
    FTypeElement = null;
}
```

Le constructeur vide initialise le type de l'ensemble.

```
public setOfObject(object[] t) {
    foreach(object elt in t)
        this.Add(elt);
}
```

Le constructeur ajoute tous les objets du tableau t dans l'ensemble.

```
public setOfObject(Array t) {
    foreach(object elt in t)
        this.Add(elt);
}
```

Le constructeur ajoute tous les objets du tableau t dans l'ensemble.

```
public setOfObject(ArrayList t) : this(t.ToArray()) {
}
```

Le constructeur convertit ArrayList t en **object[]** t et appelle le constructeur setOfObject(**object[]** t)

```
public setOfObject(string s) {
    char[] t = s.ToCharArray();
    foreach(char elt in t)
        this.Add(elt);
}
```

Le constructeur recopie les caractères de la string dans l'ensemble dont le type est donc **char**.

```
public virtual object[] ToArray() {
    object[] t = new object[this.Count];
    for(int i=0; i<this.Count; i++)
        t[i]=this[i];
    return t;
}
```

Convertit l'ensemble en tableau **object[]** (on a perdu le type).

```
public override string ToString() {
    return Convert.ToString(this.ToArray());
}
```

```
public int Card {
    get {
        return this.Count;
    }
}
```

```
protected virtual void Inserer( object sender, EventArgs e ) {
    if ( OnInserer != null )
        OnInserer( sender , e );
}
```

```
protected virtual void Enlever( object sender, EventArgs e ) {
    if ( OnEnlever != null )
        OnEnlever( sender , e );
}
```

```
public object this[ int index ] {
    get {
        return( List[index] );
    }
    set {
        List[index] = value;
    }
}
```

```

public int Add( object value ) {
    // ajoute un élément à l'ensemble sinon -1 ou -2
    if (this.Count==0)
        FTypeElement = value.GetType( );
    if (value.GetType( ) == FTypeElement) {
        if (!Contains(value) )
            return( List.Add( value ) );
        else {
            MessageBox.Show("L'élément "+value.ToString()+"", est déjà présent !",
                "Ensemble : insertion impossible (-1)");
            return -1;
        }
    }
    else {
        MessageBox.Show("L'élément "+value.ToString()+"", n'est pas du même type !",
            "Ensemble : insertion impossible (-2)");
        return -2;
    }
}

public void Remove( object value ) {
    // enlève un élément à l'ensemble s'il est présent et de même type
    if (value.GetType( ) == FTypeElement && Contains(value))
        List.Remove( value );
}

public bool Contains( object value ) {
    // true si value est dans type setOfChar, false sinon.
    return( List.Contains( value ) );
}

protected override void OnInsert( int index, Object value ) {
    Inserer( this , EventArgs.Empty );
}

protected override void OnRemove( int index, Object value ) {
    Enlever( this , EventArgs.Empty );
}
}

```

Extrait de code de test où E1, E2 et EnsCar sont des setOfObject

```

/-- chargement par ajout élément par élément :
E1.Add('a');
E1.Add('b');
E1.Add('c');
E1.Add('d');
E1.Add('e');
/-- chargement par string :
E2 = new setOfObject("xyztu#");
EnsCar = E1 + E2 ;
foreach (object elt in EnsCar)
    Console.WriteLine(" " + elt);

Console.WriteLine("type d'élément de
l'ensemble : "+EnsCar.TypeElement);

```

On teste sur le type d'élément **char**.

Contenu de l'ensemble EnsCar :

```

a
b
c
d
e
x
y
z
t
u
#
type d'élément de l'ensemble : System.Char

```

Extrait de code de test où E1, E2 et EnsCar sont des setOfObject

```
//-- chargement par ajout élément par élément :
E1.Add(45);
E1.Add(-122);
E1.Add(666);
```

```
E1.Add(45);
```

```
E1.Add(-9002211187785);
```

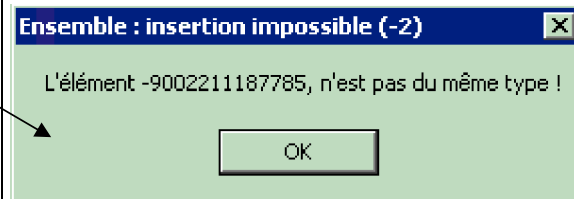
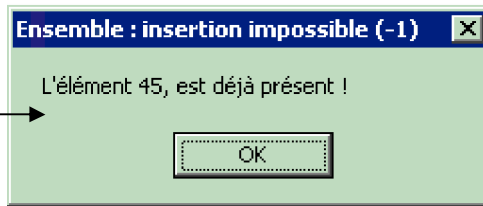
```
E1.Add(12);
```

```
//-- chargement par ajout d'un tableau d'entiers :
int[] table=new int[] { 125,126,127};
E2 = new setOfObject(table);
EnsCar = E1 + E2 ;
```

```
foreach (object elt in EnsCar)
    Console.WriteLine("  "+ elt );
```

```
Console.WriteLine("type d'élément de
l'ensemble : "+EnsCar.TypeElement);
```

On teste sur le type d'élément **int**.



-9002211187785 est de type **long** et non **int** !

```
Contenu de l'ensemble EnsCar :
45
-122
666
12
125
126
127
type d'élément de l'ensemble : System.Int32
```

classe C# de tests de la classe *setOfObject* : code source

```
class UtilisesetOfObject{

    private static void EnsbleOnEnlever( object sender, EventArgs e ) {
        Console.WriteLine("On va enlever un élément de l'ensemble");
    }

    private static void version (int choix) {
        // Déclare les ensembles
        setOfObject Ensble1 = new setOfObject();
        setOfObject EnsCar,Ensble2;

        // Initialise les ensembles selon le paramètre
        switch(choix) {
            case 1: ChargerEnsChar(Ensble1,out Ensble2); break;
            case 2: ChargerEnsInt(Ensble1,out Ensble2); break;
            case 3: ChargerEnsTChar(Ensble1,out Ensble2); break;
            case 4: ChargerEnsArrayList(Ensble1,out Ensble2); break;
            case 5: ChargerEnsArrayString(Ensble1,out Ensble2); break;
            case 6: ChargerJours(Ensble1,out Ensble2); break;
            default:ChargerEnsChar(Ensble1,out Ensble2); break;
        }
        Console.WriteLine("card(Ensble1)="+Ensble1.Card+" ");
        Console.WriteLine("card(Ensble2)="+Ensble2.Card+" ");
    }
}
```

Test d'utilisation de différents type d'éléments dans l'ensemble.

```

EnsCar=Ensble1+Ensble2;
EnsCar.OnEnlever += new EventHandler(EnsbleOnEnlever);
Console.WriteLine("card(EnsCar)="+EnsCar.Card+" ; ");
Console.WriteLine();

// Affichage du Contenu de l'ensemble avec for :
Console.WriteLine( "Contenu de l'ensemble EnsCar :");
Console.WriteLine( "===== for =====");
for(int i=0; i<EnsCar.Card; i++)
    Console.WriteLine("    "+ EnsCar[i] );

// Affichage du Contenu de l'ensemble avec foreach :
Console.WriteLine( "===== foreach =====");
foreach(object elt in EnsCar)
    Console.WriteLine("    "+ elt );

// Affichage du type d'élément de l'ensemble
Console.WriteLine("type d'élément de l'ensemble : "+EnsCar.TypeElement);
}

```

```

private static void ChargerEnsInt( setOfObject E1,out setOfObject E2) {
    //-- chargement par ajout élément par élément :
    E1.Add(45);
    E1.Add(-122);
    E1.Add(666);
    E1.Add(45);
    E1.Add(-9002211187785);
    E1.Add(12);
    //-- chargement par ajout d'un tableau d'entiers :
    int[ ] table=new int[ ] { 125,126,127 };
    E2 = new setOfObject(table);
}

```

Appel au constructeur :

public setOfObject(Array t)

```

private static void ChargerEnsChar( setOfObject E1,out setOfObject E2) {
    //-- chargement par ajout élément par élément :
    E1.Add('a');
    E1.Add('b');
    E1.Add('c');
    E1.Add('d');
    E1.Add('e');
    //-- chargement par string :
    E2 = new setOfObject("xyztu#");
}

```

Appel au constructeur :

public setOfObject(string s)

```

private static void ChargerEnsTChar( setOfObject E1,out setOfObject E2) {
    //-- chargement par ajout élément par élément :
    E1.Add('a');
    E1.Add('b');
    E1.Add('c');
    E1.Add('d');
    E1.Add('e');
    //-- chargement par tableau de char :
    char[ ] table = new char [ ] { 'x','y','z','t','u','#' };
    E2=new setOfObject(table);
}

```

Appel au constructeur :

public setOfObject(Array t)

```

private static void ChargerEnsArrayList( setOfObject E1,out setOfObject E2) {
    //-- chargement par ajout élément par élément :
    E1.Add("un");
    E1.Add("deux");
}

```

```
E1.Add("trois");
E1.Add("quatre");
E1.Add("cinq");
```

//-- chargement par ArrayList de string :

```
ArrayList t = new ArrayList( );
t.Add("six");
t.Add("sept");
t.Add("fin.");
E2=new setOfObject(t);
}
```

Appel au constructeur :

public setOfObject(ArrayList t) : **this** (t.ToArray())

qui appelle lui-même :

public setOfObject(object[] t) ←

private static void ChargerEnsArrayString(setOfObject E1, **out** setOfObject E2) {

//-- chargement par ajout élément par élément :

```
E1.Add("rat");
E1.Add("chien");
E1.Add("chat");
E1.Add("vache");
```

//-- chargement par tableau de string :

```
string[ ] table = new string[3];
table[0]="voiture";
table[1]="bus";
table[2]="fin.";
E2=new setOfObject(table);
}
```

Appel au constructeur :

public setOfObject(object[] t) ←

//--un type énuméré

```
enum Jours { Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi }
```

private static void ChargerJours(setOfObject E1, **out** setOfObject E2) {

//-- chargement par ajout élément par élément :

```
E1.Add(Jours.Vendredi);
E1.Add(Jours.Samedi);
E1.Add(Jours.Dimanche);
```

//-- chargement par tableau de TypePerso :

```
Jours[ ]table = new Jours[4];
table[0]=Jours.Lundi;
table[1]=Jours.Mardi;
table[2]=Jours.Mercredi;
table[3]=Jours.Jeudi;
E2=new setOfObject(table);
}
```

Appel au constructeur :

public setOfObject(Array t) ←

[STAThread]

public static void Main() {

```
Console.WriteLine("----- 1 -----");
version(1);
Console.WriteLine("----- 2 -----");
version(2);
Console.WriteLine("----- 3 -----");
version(3);
Console.WriteLine("----- 4 -----");
version(4);
Console.WriteLine("----- 5 -----");
version(5);
Console.WriteLine("----- 6 -----");
version(6);
}
}
```

Exécutez le programme et observez le type de l'ensemble dans chaque cas.

Classes, objet et IHM

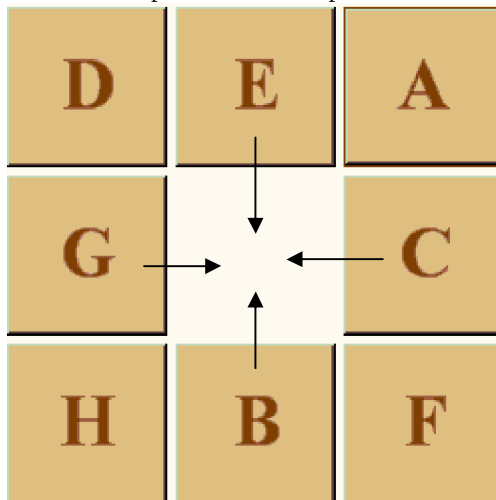
Construction d'un jeu : puzzle genre "taquin"

Soit à construire une interface de jeu du taquin permettant de ranger par ordre alphabétique, des lettres disposées dans n'importe quel ordre sur un damier 3 x 3 :

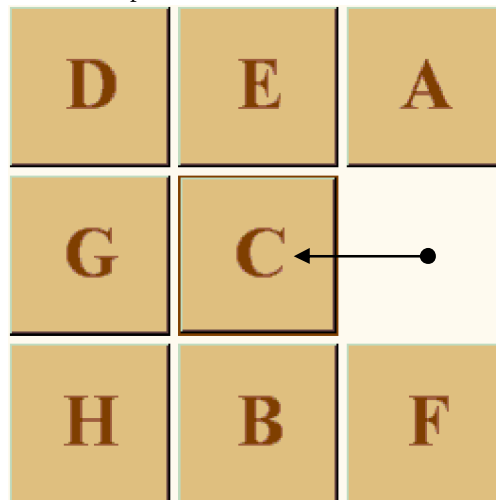


Sur les 9 cases une seule est disponible, le jeu consiste à ne déplacer qu'une seule lettre à la fois et uniquement dans la case restée libre. Par exemple dans la configuration de départ ci-haut seules les lettres G, E, C, B peuvent être déplacées vers la case centrale restée libre.

4 choix de déplacements sont possibles :



si l'on déplace la lettre C, on obtient :

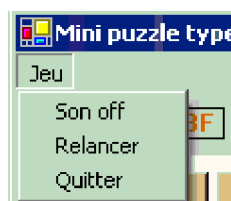


L'interface à construire en C# doit permettre à un utilisateur de jouer au taquin dans les conditions de contraintes du jeu, il utilisera la souris pour cliquer sur une lettre afin de la déplacer vers la case libre (le programme doit gérer la possibilité pour une case d'être déplacée ou non et doit tester à chaque déplacement si le joueur a gagné ou non).

L'IHM affichera la liste des lettres lue depuis le coin supérieur gauche jusqu'au coin inférieur droit du tableau (la case libre sera représentée par un #)



Lorsque le joueur a trouvé la bonne combinaison et rangé les lettres dans le bon ordre, prévoir de lui envoyer une fenêtre de dialogue de félicitation et un menu lui permettant de rejouer ou de quitter le jeu:



```

using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Runtime.InteropServices;

```

```

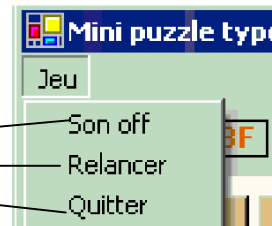
namespace WinAppliPuzzle
{
    /// <summary>
    /// Description résumée de Form1.
    /// </summary>

```

```

public class Form1 : System.Windows.Forms.Form {
    private System.Windows.Forms.MainMenu mainMenu1;
    private System.Windows.Forms.Label labelHorloge;
    private System.Windows.Forms.MenuItem menuItemjeu;
    private System.Windows.Forms.MenuItem menuItemson;
    private System.Windows.Forms.MenuItem menuItemrelancer;
    private System.Windows.Forms.MenuItem menuItemquitter;
    private System.Windows.Forms.Timer timerTemps;

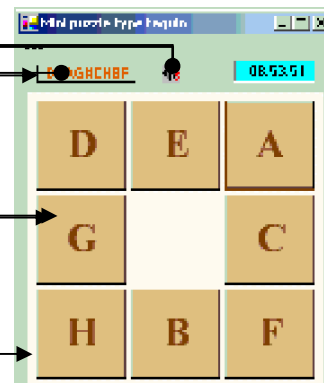
```



```

    private System.ComponentModel.IContainer components;
    private const String correct="ABCDEFGH#";
    private String modele=correct;
    private int trou=9;
    private Point Ptrou=new Point(176,176);
    private System.Windows.Forms.PictureBox pictureBoxSon;
    private System.Windows.Forms.Label labelModele;
    private System.Windows.Forms.Button buttonA;
    private System.Windows.Forms.Button buttonB;
    private System.Windows.Forms.Button buttonC;
    private System.Windows.Forms.Button buttonF;
    private System.Windows.Forms.Button buttonE;
    private System.Windows.Forms.Button buttonD;
    private System.Windows.Forms.Button buttonH;
    private System.Windows.Forms.Button buttonG;
    private System.Windows.Forms.Panel panelFond;
    private bool Sonok=false;

```



```

private int TestDeplace(int num) {
    switch (num) {
        case 1:if(trou==2 | trou==4) return trou; break;
        case 2:if(trou==1 | trou==3 | trou==5) return trou; break;
        case 3:if(trou==2 | trou==6) return trou; break;
        case 4:if(trou==1 | trou==5 | trou==7) return trou; break;
        case 5:if(trou==2 | trou==4 | trou==6 | trou==8) return trou; break;
        case 6:if(trou==3 | trou==5 | trou==9) return trou; break;
        case 7:if(trou==4 | trou==8) return trou; break;
        case 8:if(trou==5 | trou==7 | trou==9) return trou; break;
        case 9:if(trou==6 | trou==8) return trou; break;
    }
    return -1;
}

```

```

[DllImport("user32.dll",EntryPoint="MessageBeep")]
public static extern bool MessageBeep(uint uType);

```

Un exemple d'utilisation de fonction non CLS compatible et dépendant de la plateforme Win32. (envoyer un son sur le haut-parleur du PC) .

```

private void DeplaceVers(Button source, int but) {
    int T,L;
    if(but>0) {
        T = source.Location.Y;
        L = source.Location.X;
        source.Location = Prou;
        Prou = new Point(L,T);
        if (Sonok)
            MessageBeep(0);
    }
}

```

```

private void MajModele(Button source, int but) {
    if(but>0) {
        char[] s = modele.ToCharArray();
        s[(int)(source.Tag)-1]='#';
        s[but-1]=(char)(source.Text[0]);
        trou = (int)(source.Tag);
        source.Tag=but;
        modele="";
        for(int i=0; i<s.Length;i++)
            modele = modele+s[i];
    }
}

```

```

public void Tirage() {
    buttons_Click(buttonF, new EventArgs());
    buttons_Click(buttonE, new EventArgs());
    buttons_Click(buttonB, new EventArgs());
    buttons_Click(buttonA, new EventArgs());
    buttons_Click(buttonD, new EventArgs());
    buttons_Click(buttonG, new EventArgs());
    buttons_Click(buttonH, new EventArgs());
    buttons_Click(buttonB, new EventArgs());
    buttons_Click(buttonE, new EventArgs());
    buttons_Click(buttonC, new EventArgs());
    buttons_Click(buttonA, new EventArgs());
    buttons_Click(buttonE, new EventArgs());
}

```

Ecrivez une autre redistribution aléatoire des boutons. Ici nous lançons une séquence modifiant l'existant en utilisant les gestionnaires de click de souris comme si l'utilisateur avait cliqué 12 fois sur l'IHM.

```

public Form1() {
    //
    // Requis pour la prise en charge du Concepteur Windows Forms
    //
    InitializeComponent();

    //
    // TODO : ajoutez le code du constructeur après l'appel à InitializeComponent
    //
}

/// <summary>
/// Nettoyage des ressources utilisées.
/// </summary>
protected override void Dispose( bool disposing ) {
    if( disposing ) {
        if (components != null) {
            components.Dispose();
        }
    }
}

```

```

base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// <summary>
/// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
/// le contenu de cette méthode avec l'éditeur de code.
/// </summary>
private void InitializeComponent()
{
    this.components = new System.ComponentModel.Container();
    System.Resources.ResourceManager resources = new System.Resources.ResourceManager(typeof(Form1));
    this.mainMenu1 = new System.Windows.Forms.MainMenu();
    this.menuItemjeu = new System.Windows.Forms.MenuItem();
    this.menuItemson = new System.Windows.Forms.MenuItem();
    this.menuItemrelancer = new System.Windows.Forms.MenuItem();
    this.menuItemquitter = new System.Windows.Forms.MenuItem();
    this.panelFond = new System.Windows.Forms.Panel();
    this.buttonH = new System.Windows.Forms.Button();
    this.buttonG = new System.Windows.Forms.Button();
    this.buttonF = new System.Windows.Forms.Button();
    this.buttonE = new System.Windows.Forms.Button();
    this.buttonD = new System.Windows.Forms.Button();
    this.buttonC = new System.Windows.Forms.Button();
    this.buttonB = new System.Windows.Forms.Button();
    this.buttonA = new System.Windows.Forms.Button();
    this.pictureBoxSon = new System.Windows.Forms.PictureBox();
    this.labelModele = new System.Windows.Forms.Label();
    this.timerTemps = new System.Windows.Forms.Timer(this.components);
    this.labelHorloge = new System.Windows.Forms.Label();
    this.panelFond.SuspendLayout();
    this.SuspendLayout();
    //
    // mainMenu1
    //
    this.mainMenu1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[]
    {
        this.menuItemjeu
    }
    );
    //
    // menuItemjeu
    //
    this.menuItemjeu.Index = 0;
    this.menuItemjeu.MenuItems.AddRange(new System.Windows.Forms.MenuItem[]
    {
        this.menuItemson,
        this.menuItemrelancer,
        this.menuItemquitter
    }
    );
    this.menuItemjeu.Text = "Jeu";
    //
    // menuItemson
    //
    this.menuItemson.Index = 0;
    this.menuItemson.Text = "Son off";
    this.menuItemson.Click += new System.EventHandler(this.menuItemson_Click);
    //
    // menuItemrelancer

```

```

//
this.menuItemrelancer.Index = 1;
this.menuItemrelancer.Text = "Relancer";
this.menuItemrelancer.Click += new System.EventHandler(this.menuItemrelancer_Click);
//
// menuItemquitter
//
this.menuItemquitter.Index = 2;
this.menuItemquitter.Text = "Quitter";
this.menuItemquitter.Click += new System.EventHandler(this.menuItemquitter_Click);
//
// panelFond
//
this.panelFond.BackColor = System.Drawing.SystemColors.Info;
this.panelFond.Controls.Add(this.buttonH);
this.panelFond.Controls.Add(this.buttonG);
this.panelFond.Controls.Add(this.buttonF);
this.panelFond.Controls.Add(this.buttonE);
this.panelFond.Controls.Add(this.buttonD);
this.panelFond.Controls.Add(this.buttonC);
this.panelFond.Controls.Add(this.buttonB);
this.panelFond.Controls.Add(this.buttonA);
this.panelFond.ImeMode = System.Windows.Forms.ImeMode.NoControl;
this.panelFond.Location = new System.Drawing.Point(8, 32);
this.panelFond.Name = "panelFond";
this.panelFond.Size = new System.Drawing.Size(264, 264);
this.panelFond.TabIndex = 0;
//
// buttonH
//
this.buttonH.BackColor = System.Drawing.Color.Tan;
this.buttonH.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.buttonH.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonH.Location = new System.Drawing.Point(92, 176);
this.buttonH.Name = "buttonH";
this.buttonH.Size = new System.Drawing.Size(80, 80);
this.buttonH.TabIndex = 16;
this.buttonH.Tag = 8;
this.buttonH.Text = "H";
this.buttonH.Click += new System.EventHandler(this.buttons_Click);
this.buttonH.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonG
//
this.buttonG.BackColor = System.Drawing.Color.Tan;
this.buttonG.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.buttonG.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonG.Location = new System.Drawing.Point(8, 176);
this.buttonG.Name = "buttonG";
this.buttonG.Size = new System.Drawing.Size(80, 80);
this.buttonG.TabIndex = 15;
this.buttonG.Tag = 7;
this.buttonG.Text = "G";
this.buttonG.Click += new System.EventHandler(this.buttons_Click);
this.buttonG.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonF
//

```

```

this.buttonF.BackColor = System.Drawing.Color.Tan;
this.buttonF.Cursor = System.Windows.Forms.Cursors.Default;
this.buttonF.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.buttonF.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonF.Location = new System.Drawing.Point(176, 92);
this.buttonF.Name = "buttonF";
this.buttonF.Size = new System.Drawing.Size(80, 80);
this.buttonF.TabIndex = 14;
this.buttonF.Tag = 6;
this.buttonF.Text = "F";
this.buttonF.Click += new System.EventHandler(this.buttons_Click);
this.buttonF.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonE
//
this.buttonE.BackColor = System.Drawing.Color.Tan;
this.buttonE.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.buttonE.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonE.Location = new System.Drawing.Point(92, 92);
this.buttonE.Name = "buttonE";
this.buttonE.Size = new System.Drawing.Size(80, 80);
this.buttonE.TabIndex = 13;
this.buttonE.Tag = 5;
this.buttonE.Text = "E";
this.buttonE.Click += new System.EventHandler(this.buttons_Click);
this.buttonE.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonD
//
this.buttonD.BackColor = System.Drawing.Color.Tan;
this.buttonD.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.buttonD.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonD.Location = new System.Drawing.Point(8, 92);
this.buttonD.Name = "buttonD";
this.buttonD.Size = new System.Drawing.Size(80, 80);
this.buttonD.TabIndex = 12;
this.buttonD.Tag = 4;
this.buttonD.Text = "D";
this.buttonD.Click += new System.EventHandler(this.buttons_Click);
this.buttonD.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonC
//
this.buttonC.BackColor = System.Drawing.Color.Tan;
this.buttonC.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.buttonC.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonC.Location = new System.Drawing.Point(176, 8);
this.buttonC.Name = "buttonC";
this.buttonC.Size = new System.Drawing.Size(80, 80);
this.buttonC.TabIndex = 11;
this.buttonC.Tag = 3;
this.buttonC.Text = "C";
this.buttonC.Click += new System.EventHandler(this.buttons_Click);
this.buttonC.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonB

```

```

//
this.buttonB.BackColor = System.Drawing.Color.Tan;
this.buttonB.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)0));
this.buttonB.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonB.Location = new System.Drawing.Point(92, 8);
this.buttonB.Name = "buttonB";
this.buttonB.Size = new System.Drawing.Size(80, 80);
this.buttonB.TabIndex = 10;
this.buttonB.Tag = 2;
this.buttonB.Text = "B";
this.buttonB.Click += new System.EventHandler(this.buttons_Click);
this.buttonB.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// buttonA
//
this.buttonA.BackColor = System.Drawing.Color.Tan;
this.buttonA.Font = new System.Drawing.Font("Times New Roman", 27.75F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)0));
this.buttonA.ForeColor = System.Drawing.Color.SaddleBrown;
this.buttonA.Location = new System.Drawing.Point(8, 8);
this.buttonA.Name = "buttonA";
this.buttonA.Size = new System.Drawing.Size(80, 80);
this.buttonA.TabIndex = 9;
this.buttonA.Tag = 1;
this.buttonA.Text = "A";
this.buttonA.Click += new System.EventHandler(this.buttons_Click);
this.buttonA.MouseMove += new System.Windows.Forms.MouseEventHandler(this.buttons_MouseMove);
//
// pictureBoxSon
//
this.pictureBoxSon.Image = ((System.Drawing.Image)(resources.GetObject("pictureBoxSon.Image")));
this.pictureBoxSon.Location = new System.Drawing.Point(128, 8);
this.pictureBoxSon.Name = "pictureBoxSon";
this.pictureBoxSon.Size = new System.Drawing.Size(16, 16);
this.pictureBoxSon.SizeMode = System.Windows.Forms.PictureBoxSizeMode.AutoSize;
this.pictureBoxSon.TabIndex = 1;
this.pictureBoxSon.TabStop = false;
//
// labelModele
//
this.labelModele.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
this.labelModele.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)0));
this.labelModele.ForeColor = System.Drawing.Color.Chocolate;
this.labelModele.Location = new System.Drawing.Point(16, 8);
this.labelModele.Name = "labelModele";
this.labelModele.Size = new System.Drawing.Size(88, 16);
this.labelModele.TabIndex = 2;
this.labelModele.Text = "ABCDEFGH#";
this.labelModele.TextAlign = System.Drawing.ContentAlignment.TopCenter;
//
// timerTemps
//
this.timerTemps.Enabled = true;
this.timerTemps.Interval = 1000;
this.timerTemps.Tick += new System.EventHandler(this.timer1_Tick);
//
// labelHorloge
//

```

```

this.labelHorloge.BackColor = System.Drawing.Color.Aqua;
this.labelHorloge.BorderStyle = System.Windows.Forms.BorderStyle.Fixed3D;
this.labelHorloge.Font = new System.Drawing.Font("Microsoft Sans Serif", 8.25F,
    System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point, ((byte)(0)));
this.labelHorloge.Location = new System.Drawing.Point(192, 4);
this.labelHorloge.Name = "labelHorloge";
this.labelHorloge.Size = new System.Drawing.Size(72, 20);
this.labelHorloge.TabIndex = 4;
this.labelHorloge.Text = "00:00:00";
this.labelHorloge.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(5, 13);
this.ClientSize = new System.Drawing.Size(280, 305);
this.Controls.Add(this.labelHorloge);
this.Controls.Add(this.labelModele);
this.Controls.Add(this.pictureBoxSon);
this.Controls.Add(this.panelFond);
this.Menu = this.mainMenu1;
this.Name = "Form1";
this.Text = "Mini puzzle type taquin";
this.Load += new System.EventHandler(this.Form1_Load);
this.panelFond.ResumeLayout(false);
this.ResumeLayout(false);
}
#endregion

```

```

/// <summary>
/// Point d'entrée principal de l'application.
/// </summary>
[STAThread]
static void Main() {
    Application.Run(new Form1());
}

```

**Gestionnaire centralisé du
click de souris sur les 8
boutons représentant les
lettres dans le tableau.**

```

private void buttons_Click(object sender, EventArgs e) {
    int but = TestDeplace((int)((sender as Button).Tag));
    DeplaceVers((sender as Button), but);
    MajModele((sender as Button), but);
    labelModele.Text = modele;
    if(modele.CompareTo(correct)==0)
        MessageBox.Show("Bravo vous avez trouvé !", "Information",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
}

```

```

private void timer1_Tick(object sender, EventArgs e) {
    DateTime dt = DateTime.Now;
    labelHorloge.Text=dt.ToString("T");
}

```

**Gestionnaire centralisé du
Move de souris sur les 8
boutons représentant les
lettres dans le tableau.**

```

private void buttons_MouseMove(object sender, System.Windows.Forms.MouseEventArgs e) {
    if (TestDeplace((int)((sender as Button).Tag))>0)
        (sender as Button).Cursor = System.Windows.Forms.Cursors.Hand;
    else
        (sender as Button).Cursor = System.Windows.Forms.Cursors.No;
}

```

//-- les gestionnaires des click dans le menu :

```

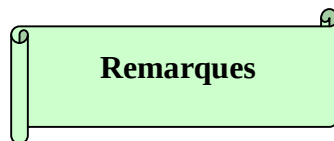
private void menuItemson_Click(object sender, System.EventArgs e) {
    Console.WriteLine(menuItemson.Text);
    if(menuItemson.Text.IndexOf("off")>0) {
        menuItemson.Text = "Son on";
        Sonok = false;
        pictureBoxSon.Visible = false;
    }
    else {
        menuItemson.Text = "Son off";
        Sonok = true;
        pictureBoxSon.Visible = true;
    }
}

private void menuItemrelancer_Click(object sender, System.EventArgs e) {
    Tirage();
}

private void menuItemquitter_Click(object sender, System.EventArgs e) {
    Close();
}

private void Form1_Load(object sender, System.EventArgs e) {
    Tirage();
    Sonok = true;
}
}

```



Nous avons utilisé la classe **System.Windows.Forms.Timer** afin d'afficher l'heure système dans le jeu :

System.Object

```

|__ System.MarshalByRefObject
    |__ System.ComponentModel.Component
        |__ System.Windows.Forms.Timer

```

Cette classe très semblable à la classe TTimer de Delphi, implémente une minuterie déclenchant un événement **Tick** (événement OnTimer en Delphi) selon un intervalle défini par l'utilisateur (préconisation Microsoft : un Timer doit être utilisé dans une fenêtre). Ci-dessous le gestionnaire de l'événement **Tick** :

```

private void timer1_Tick(object sender, System.EventArgs e) {
    DateTime dt = DateTime.Now;
    labelHorloge.Text=dt.ToString("T");
}

```

Récupère la date du jour (sous la forme d'un entier)

System.Object

```

|__ System.ValueType
    |__ System.DateTime

```

Converti l'entier au format :
Hh : mm : ss

Les valeurs de date de type **DateTime** sont mesurées en unités de 100 nanosecondes et exprimées sous forme d'un entier long.

Remarque concernant l'instruction chargeant une image :

```
this.pictureBoxSon.Image = ((System.Drawing.Image)(resources.GetObject("pictureBoxSon.Image")));
```

L'image utilisée sous le nom "pictureBoxSon.Image" doit figurer dans un fichier de ressource (ou y être ajoutée) suivant la manière dont le projet est construit (projet vide, application windows ...). L'instruction précédente permet alors l'accès à l'image "pictureBoxSon.Image" par l'accès au fichier de ressource.