

Listes et structures



Objectifs

- Comprendre la notion de liste
- Utilisation des listes en C#
- Comprendre la notion de structure
- Utiliser les classes C# pour les implanter

Tableau : problèmes

- Un tableau a une capacité fixe
- Comment représenter une collection de valeurs sans connaître au préalable leur nombre ?
 - Déclarer avec un tableau qui a une capacité de 0
 - À chaque ajout de valeur :
 1. Déclarer un tableau avec une capacité égale à la capacité précédente + 1
 2. Recopier le premier tableau dans le nouveau
 3. Placer la nouvelle valeur dans le nouveau tableau
 4. Utiliser le nouveau tableau

⇒ C'est fastidieux !

Les listes

- Comme les tableaux, les listes permettent de représenter des collections de valeurs d'un certain type
- Une liste se comporte presque comme un tableau, mais diffère par :
 - Un tableau a une capacité, une liste a un nombre de valeurs
 - Une liste permet l'ajout de valeurs sans se soucier de la capacité de la collection

Les listes déclarations

- Déclaration et initialisation :

`Liste<<type>> <nomVariable> = créer Liste<<type>>();`

ou

`Liste<<type>> <nomVariable> = null;`

- Exemple :

`Liste<réel> notesEtudiants = créer Liste<réel>();`

⇒ Déclaration d'une variable de type référence, une liste de réels

⇒ Initialisation de la liste avec 0 élément

- Obtenir le nombre d'éléments :

– `<nomVariable>.Quantité`

Les listes – Manipulation

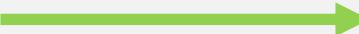
- Pour ajouter une valeur, on utilise l'instruction
".Ajouter(<valeur>)"
- Pour supprimer une valeur, on utilise l'instruction
".Supprimer(<indice>)"
- Pour accéder à une valeur, on utilise l'opérateur « [] » en lui donnant l'indice dont on veut la valeur :
<variableTypeTableau>[<indice>] : agit comme une variable que vous avez l'habitude de manipuler

Les listes exemple 1/3

```
Liste<réel> notesEtudiants = créer Liste<réel>();  
notesEtudiants.Ajouter(78);  
notesEtudiants.Ajouter(67);  
notesEtudiants.Ajouter(42);  
notesEtudiants.Ajouter(98); // État 1  
  
notesEtudiants.Supprimer(1); // État 2  
  
notesEtudiants[1] = 99; // État 3  
  
écrireNL("La première note est : " + notesEtudiants[0].VersChaine());
```

notesEtudiants // État 1

{ }



0	1	2	3
78	67	42	98

Les listes exemple 2/3

```
Liste<réel> notesEtudiants = créer Liste<réel>();  
notesEtudiants.Ajouter(78);  
notesEtudiants.Ajouter(67);  
notesEtudiants.Ajouter(42);  
notesEtudiants.Ajouter(98); // État 1  
  
notesEtudiants.Supprimer(1); // État 2  
  
notesEtudiants[1] = 99; // État 3  
  
écrireNL("La première note est : " + notesEtudiants[0].VersChaine());
```

notesEtudiants // État 2

0	1	2	3
78	67	42	98



0	1	2
78	42	98

Les listes exemple 3/3

```
Liste<réel> notesEtudiants = créer Liste<réel>();  
notesEtudiants.Ajouter(78);  
notesEtudiants.Ajouter(67);  
notesEtudiants.Ajouter(42);  
notesEtudiants.Ajouter(98); // État 1  
  
notesEtudiants.Supprimer(1); // État 2  
  
notesEtudiants[1] = 99; // État 3  
  
écrireNL("La première note est : " + notesEtudiants[0].VersChaine());
```

notesEtudiants // État 2

0	1	2
78	42	98



0	1	2
78	99	98

La première note est : 78 ↵

Et en C# ?

- Déclaration et initialisation :

```
List<<type>> <nomVariable> = new List<<type>>();
```

ou

```
List<<type>> <nomVariable> = null;
```

- Exemple :

```
List<decimal> notesEtudiants = new List<decimal>();
```

⇒ Déclaration d'une variable de type référence, une liste de réels

⇒ Initialisation de la liste avec 0 élément

- Obtenir le nombre d'éléments :

– <nomVariable>.Count

Et en C# ? Suite

- Pour ajouter une valeur, on utilise l'instruction
".Add(<valeur>)"
- Pour supprimer une valeur, on utilise l'instruction
".RemoveAt(<indice>)"
- Pour accéder à une valeur, on utilise l'opérateur « [] » en lui donnant l'indice dont on veut la valeur :
<variableTypeTableau>[<indice>] : agit comme une variable que vous avez l'habitude de manipuler

L'algorithme précédent devient

```
Liste<réel> notesEtudiants = créer Liste<réel>();  
notesEtudiants.Ajouter(78);  
notesEtudiants.Ajouter(67);  
notesEtudiants.Ajouter(42);  
notesEtudiants.Ajouter(98);  
  
notesEtudiants.Supprimer(1);  
  
notesEtudiants[1] = 99;  
  
écrireNL("La première note est : " +  
notesEtudiants[0].VersChaine());
```

```
List<decimal> notesEtudiants = new List<decimal>();  
notesEtudiants.Add(78);  
notesEtudiants.Add(67);  
notesEtudiants.Add(42);  
notesEtudiants.Add(98);  
  
notesEtudiants.RemoveAt(1);  
  
notesEtudiants[1] = 99;  
  
Console.Out.WriteLine("La première note est : " +  
notesEtudiants[0].ToString());
```

Structures

- Une structure est un type créé par l'utilisateur qui a un nom et qui permet le **regroupement de données** sous la forme de champs
- Chaque champ est :
 - Identifié par un nom
 - D'un certain type
 - **Le type peut lui-même être une structure**
- Une fois définie, une structure **s'utilise comme n'importe quel type**

Structures – Définition

- Déclaration :

```
structure <nomStructure> {  
    <type> <nomChamp1>;  
    [...]  
    <type> <nomChampN>;  
}
```

- Exemple :

```
structure LigneFacture {  
    chaîne description;  
    entier quantite;  
    réel prixUnitaire;  
}
```

Structures – Utilisation 1/2

- Déclaration :

```
<nomStructure> <nomVariable> = null;
```

ou

```
<nomStructure> <nomVariable> = créer <nomStructure>();
```

- Exemple :

```
LigneFacture premiereLigne = créer LigneFacture();
```

Structures – Utilisation 2/2

- Pour accéder à un champ, il faut utiliser la notation ".":
<nomVariable>.<nomChamp>;
- Exemple :

```
LigneFacture premiereLigne = créer LigneFacture();
premiereLigne.description = "Clean code";
premiereLigne.quantite = 5;
premiereLigne.prixUnitaire = 39.90;

écrireNL("Le prix unitaire est : " + premiereLigne.prixUnitaire.VersChaine());
```

Structures – C#

- Dans ce cours, les structures seront implantées par des classes
- L'utilisation des classes sera donc limitée à la représentation de structures (Les autres notions seront abordées en programmation orientée objet)
- Les classes sont des types référence
- Les champs seront implantés par des auto-propriétés

Classes – C#

- On ajoute une fichier à partir du gabarit Visual Studio « classe »
- On indique « **public** » devant le nom de la classe

```
public class LigneFacture
{
    // Propriétés (Champs de la classe)
}
```

Propriétés – C#

- La déclaration est codée comme suit :

```
public <type> <nom propriété> { get; set; }
```

```
Exemple : public string Description { get; set; }
```

- Dans Visual Studio, utilisez le fragment (*snippet*) « prop »
- Exemple :

```
public class LigneFacture // Fichier LigneFacture.cs
{
    public string Description { get; set; }
    public int Quantite { get; set; }
    public decimal PrixUnitaire { get; set; }
}
```

Traduction du pseudocode en C#

```
LigneFacture premiereLigne = créer LigneFacture();
premiereLigne.description = "Clean code";
premiereLigne.quantite = 5;
premiereLigne.prixUnitaire = 39.90;

écrire("Le prix unitaire est : ");
écrireNL(premiereLigne.prixUnitaire);
```

```
LigneFacture premiereLigne = new LigneFacture();
premiereLigne.Description = "Clean code";
premiereLigne.Quantite = 5;
premiereLigne.PrixUnitaire = 39.90m;

Console.Out.Write("Le prix unitaire est : ");
Console.Out.WriteLine(premiereLigne.PrixUnitaire);
```

Tests unitaires et structures

- Pour vos tests unitaires, vous ne pourrez pas tester l'égalité de deux valeurs de type structure : vous allez devoir tester chaque champ de la structure