(Leçon 02)
Structures de données en C#

LOTFI DERBALI, PH.D
DERBALI.IT@GMAIL.COM
HTTPS://GITHUB.COM/DERBALI-IT

Contenu

- Structures de données élémentaires
  - Listes
  - Listes chainées
  - Piles
  - Files
- Structures de données avancées
  - table de hachage
  - Dictionnaire

Types

- Un type est un ensemble (possiblement infini) de valeurs et d'opérations sur celles-ci
  - En C#
    - Types primitifs (int, float, bool, …)
    - Types agrégés (tableaux et ceux définis par les classes)
- Un type abstrait (TA) est un type accessible uniquement a travers une interface
  - Exemple: le type de dictionnaire ou table de symboles représente un ensemble d'associations (clé; info) avec clés uniques.
    - search(k)
    - add(clé; info)

- Une structure de données est une manière d'organiser et de stocker l'information
- Une structure de données a une interface qui consiste en un ensemble de procédures pour ajouter, effacer, accéder, réorganiser, etc. les données.
- Exemples de structures de données
  - Pile : collection d'objets accessible selon une politique LIFO
  - File : collection d'objets accessible selon une politique FIFO
  - File double : combine accès LIFO et FIFO
  - Liste : collection d'objets ordonnés accessible à partir de leur position
  - Vecteur : collection d'objets ordonnés accessible à partir de leur rang
  - File à priorité : accès uniquement à l'élément de clé (priorité) maximale

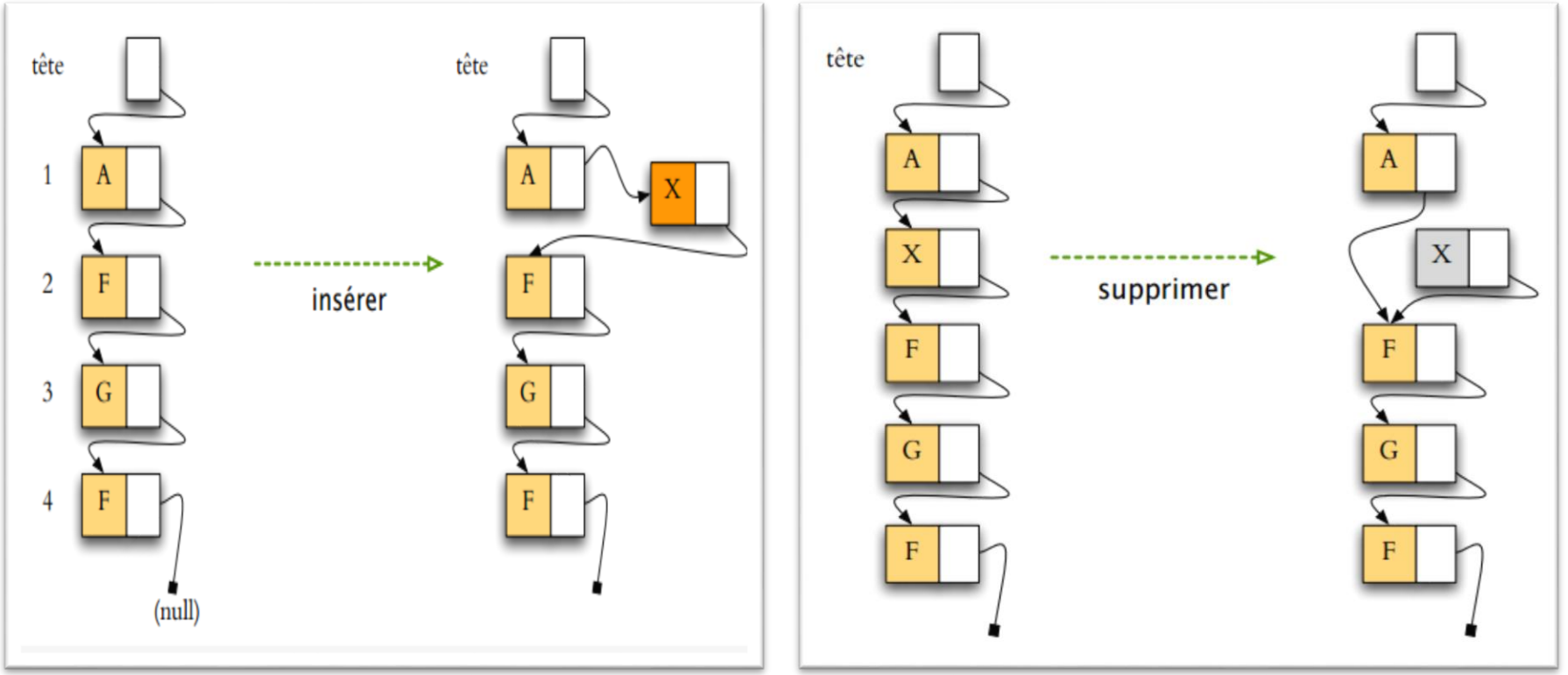
Les Listes

- Classe générique List<T>
  - située dans l'espace de noms System.Collections.Generic
  - Exemple: List<string>
- Comparaison avec les tableaux
  - la taille d'un tableau est fixe alors que la taille d'une liste est variable

```
List<string> firstNameList = new List<string>();
// Ajoute des éléments.
firstNameList.Add("Matt");
firstNameList.Add("Tim");
firstNameList.Add("James");

Console.WriteLine(firstNameList.Count);

// Remove retourne true si l'élément a été supprimé et false sinon.
if (firstNameList.Remove("Tim"))
{
    Console.WriteLine("Tim a bien été supprimé de la liste.");
}
else
{
    Console.WriteLine("Tim n'a pas été supprimé de la liste.");
}

firstNameList.Clear();
```

Les listes chainées (LinkedList)

- située dans l'espace de noms System.Collections
- Un ensemble d'éléments conservés chacun dans un nœud contenant un ou deux liens sur le nœud suivant et/ou précédent dans la liste
- Insertion et suppression



Les listes chainées

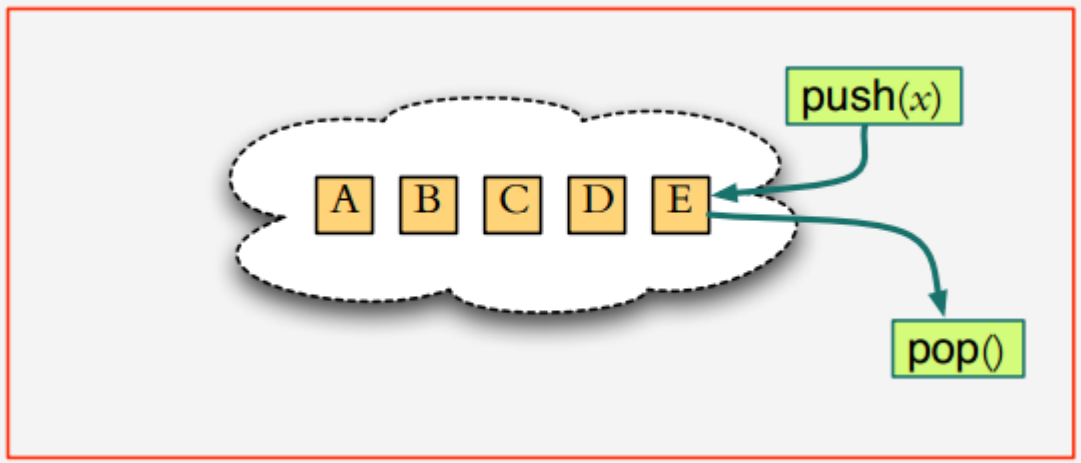
- Un ensemble d'éléments conservés chacun dans un nœud contenant un ou deux liens sur le nœud suivant et/ou précédent dans la liste
- Exemple

```
// Créer et initialiser une liste chaînée.
LinkedList<String> ll = new LinkedList<String>();
ll.AddLast("rouge");
ll.AddLast("orange");
ll.AddLast("jaune");
ll.AddLast("orange");

// Afficher le contenu de la liste chaînée.
if (ll.Count > 0)
{
    Console.WriteLine("   Le premier noeud dans la liste est {0}.", ll.First.Value);
    Console.WriteLine("   Le dernier noeud dans la liste est {0}.", ll.Last.Value);

    Console.WriteLine("   La liste chaînée contient:");
    foreach (String s in ll)
        Console.WriteLine("     {0}", s);
}
else
{
    Console.WriteLine("   La liste chaînée est vide.");
}
```

Les piles

- Dans une pile (Stack), l'élément le plus récemment ajouté est celui qui est retiré avant les autres (last in first out)
- Les opérations de base sont push («empiler») et pop («dépiler»).

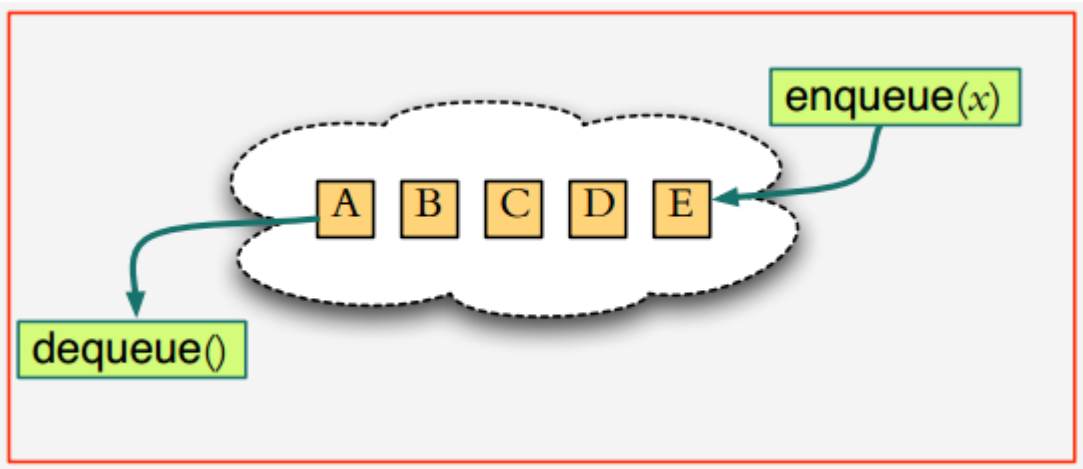
```
// Créer and initialier une pile.
Stack<String> myStack = new Stack<String>();
myStack.Push("La");
myStack.Push("stucture");
myStack.Push("de");
myStack.Push("données");

// Afficher le contenu de la pile.
Console.WriteLine("Valeurs de la pile: ");
while (myStack.Count > 0)
    Console.WriteLine("   {0}", myStack.Pop());
```



Les files simples

- Dans une queue ou file FIFO, l'élément le plus ancien sera retiré avant les autres (first in first out)
- Les opérations de base sont enqueue («enfiler») et dequeue («défiler»).

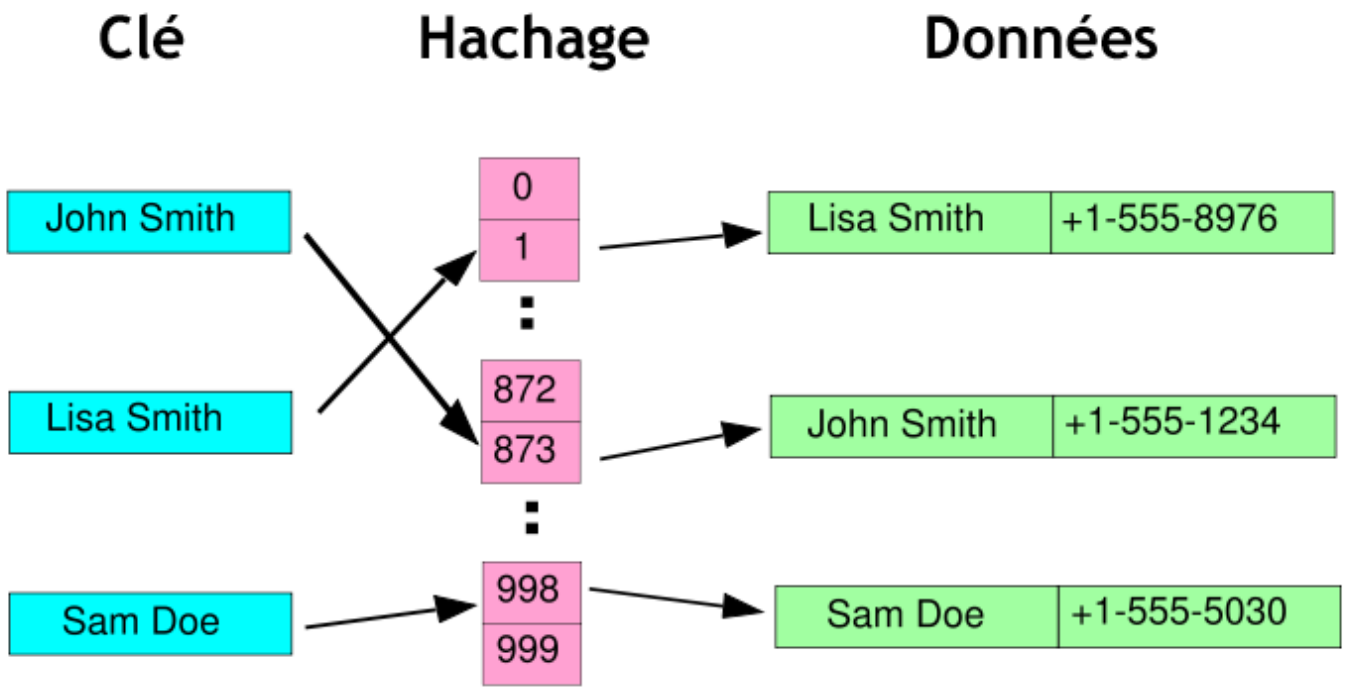
```
// Créer and initialier une file FIFO.
Queue<String> myQueue = new Queue<String>();
myQueue.Enqueue("La");
myQueue.Enqueue("stucture");
myQueue.Enqueue("de");
myQueue.Enqueue("données");

// Afficher le contenu de la pile.
Console.WriteLine("   Valeurs de la file FIFO: ");
while (myQueue.Count > 0)
    Console.WriteLine("      {0}", myQueue.Dequeue());
```



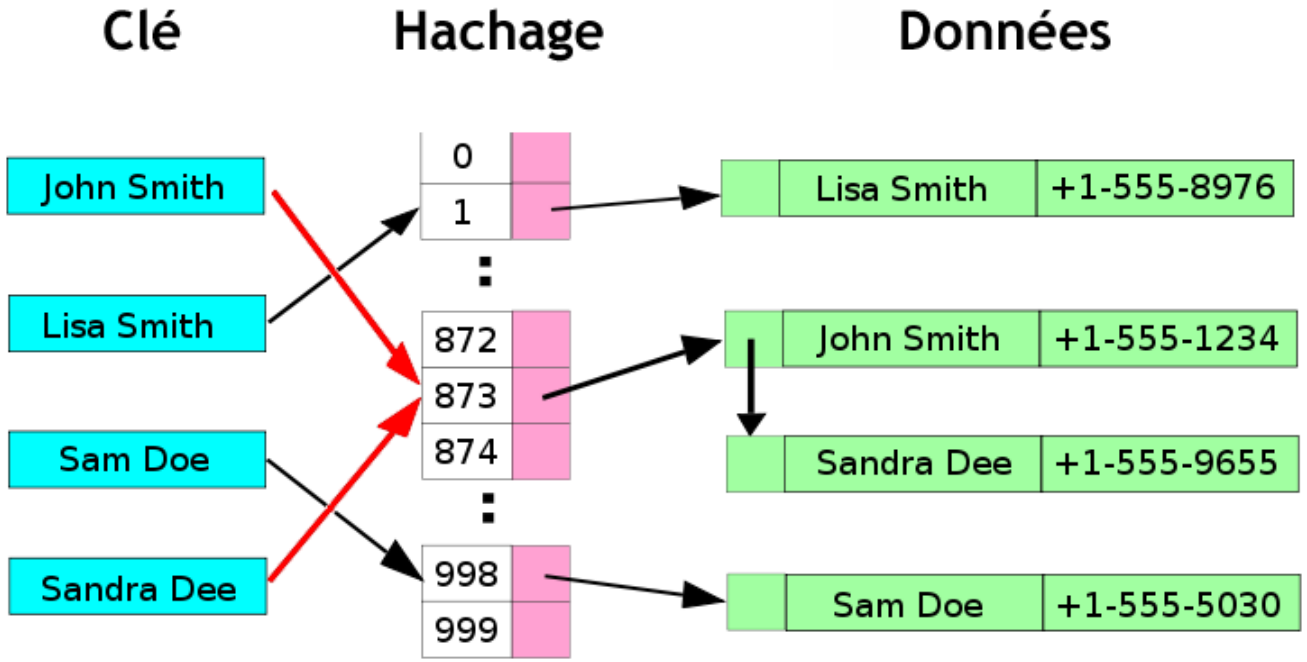
Structures de données avancées

- Table de hachage
  - Une table de hachage (*hash table* en anglais) est une structure de donnée permettant d'associer une valeur à une clé.
  - Il s'agit d'un tableau ne comportant pas d'ordre.
  - Exemples:
    - On peut imaginer un répertoire dans un téléphone portable : la clé serait le nom du contact, tandis que la valeur serait le numéro de téléphone.
    - Un dictionnaire, où vous associez un mot (clé) à une définition (valeur).

- L'accès à un élément se fait en transformant la clé en une valeur de hachage par l'intermédiaire d'une fonction de hachage.
- Le hachage est un nombre qui permet la localisation des éléments dans le tableau.



- Lorsque deux clés ont la même valeur de hachage, ces clés ne peuvent être stockées à la même position, on doit alors employer une stratégie de résolution des collisions.
  - Chaque case de la table est en fait une liste chaînée des clés qui ont le même hachage.



- Exemple
  - Insertion dans une table de hachage
  - Lecture des clés et ses valeurs associées

```
// Créer and initialier une table de hachage.
Hashtable myHT = new Hashtable();
myHT.Add("un", "une");
myHT.Add("deux", "structure");
myHT.Add("trois", "de");
myHT.Add("quatre", "données");

// Affihcer le contenu de la table de hachage.
Console.WriteLine("La table de hachage contient:");
foreach (DictionaryEntry de in myHT)
    Console.WriteLine("\t{0}:\t{1}", de.Key, de.Value);
Console.WriteLine();
```

Les collections et leurs relations avec les interfaces

- En C#, les collections ont été créées à partir des interfaces.

```
public class List<T> : IList<T>, ICollection<T>, IList, ICollection,
                       IReadOnlyList<T>, IReadOnlyCollection<T>, IEnumerable<T>,
                       IEnumerable
```

Interfaces	Description
IList	Représente une collection d'objets accessibles séparément par index.
ICollection	Définit des méthodes pour manipuler des collections génériques.
IReadOnlyList	Représente une collection en lecture seule d'éléments accessibles par index.
IReadOnlyCollection	Représente une collection fortement typée, en lecture seule des éléments.
IEnumerable	Expose l'énumérateur, qui prend en charge une itération simple sur une collection non générique.

LINQ (Language INtegrated Query)

- Outil puissant pour accéder, récupérer et transformer de données
- LINQ permet d'effectuer des requêtes directement en langage C# sans passer par un langage secondaire, tel que SQL ou XQuery
  - Bases de données SQL, Structures de données en mémoire, documents XML, etc.
- La syntaxe des expressions de requête est similaire à celle de SQL

- Mots-clés
  - Le mot-clé « select » définit les éléments de la requête qui seront sélectionnés et accessibles par la suite
  - Le mot-clé « new » établit la requête de manière plus personnalisée. L'élément ainsi créé retournera des objets anonymes dans lesquels seront rangés les éléments de données
  - Le mot-clé « where » qui prend en argument une expression conditionnelle qui retournera un booléen

Création d'une requête LINQ

Mots clés	Description
from	Introduit la variable d'itération (ou l'énumérateur) qui sert à parcourir la collection
where	Spécifie la condition qui teste les éléments de la collection (une expression booléenne);
select	Définit les données retournées en une collection créée par LINQ.

- Exemple: Parmi la liste de prénoms, extraire seulement ceux de 4 caractères.

```
// Requête: Parmi la liste de prénoms, extraire seulement ceux de 4 car.
IEnumerable<string> req = from sPrenom in prenoms
                          where sPrenom.Length == 4
                          select sPrenom;
```

Requête simple

- une simple requête LINQ concernant une solution logicielle qui retourne les noms de clients en Italie

```
var query =
    from   c in Customers
    where  c.Country == "Italy"
    select c.CompanyName;
```

- Le résultat de cette requête est une liste de chaînes

```
foreach ( string name in query ) {
    Console.WriteLine( name );
}
```

- La méthode ToList produit une collection de List<T>

```
var query = from c in Customers ...
List<Customer> customers = query.ToList();
```

Méthodes de calculs groupés

- LINQ implémente des mots servant à retourner directement des valeurs particulières d'une liste de la même manière qu'en SQL

Méthodes	Description
Min()	Permet d'obtenir la valeur minimum sur une liste de valeurs sélectionnées.
Max()	Permet d'obtenir la valeur maximum sur une liste de valeurs sélectionnées.
Count()	Permet de connaitre le nombre d'éléments sélectionnés.
Average()	Permet de connaitre la moyenne de valeurs.
Sum()	Réalise la somme de valeurs.
Aggregate()	Permet de créer sa propre valeur à partir de celles sélectionnées.

- Exemples:

```
// Créer et initialiser une liste chaînée.
List<String> myList = new List<String>();
myList.Add("rouge");
myList.Add("orange");
myList.Add("jaune");
myList.Add("orange");

// Requête LINQ.
var requete1 = from s in myList
               where s.Equals("orange")
               select s;

// Affihcer le résultat de la requête.
Console.Write("Il y a "+requete1.Count()+" objets \"orange\"\n");
```

```
// Créer et initialiser une liste chaînée.
List<String> myList = new List<String>();
myList.Add("rouge");
myList.Add("orange");
myList.Add("jaune");
myList.Add("orange");

// Requête LINQ.
var requete1 = from s in myList
               select new { texte = s, valBool = s.Contains("g") };

// Affihcer le résultat de la requête.
foreach (var v in requete1)
{
    Console.WriteLine("L'objet \"" + v.texte + "\" contient la lettre \"g\" : " + v.valBool);
}
```

Trier avec LINQ

Mot clé	Description
orderby	La clause orderby entraîne le tri en ordre croissant ou décroissant de la séquence ou de la sous-séquence (groupe) retournée dans une expression de requête. Plusieurs clés peuvent être spécifiées pour effectuer une ou plusieurs opérations de tri secondaires. Le tri est effectué par le comparateur par défaut pour le type de l'élément. L'ordre de tri par défaut est le tri croissant.

Ordre	Description
ascending	Le mot clé contextuel ascending est utilisé dans la clause orderby dans les expressions de requête afin de spécifier que l'ordre de tri est **du plus petit au plus grand**. **(DÉFAUT)**
descending	Le mot clé contextuel descending est utilisé dans la clause orderby dans les expressions de requête afin de spécifier que l'ordre de tri est **du plus grand au plus petit**.

- Exemple:

```
var req = from sPrenom in _prenoms
          where sPrenom.Length == 4
          orderby sPrenom descending
          select sPrenom;
```