

# *STL*

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Introduction

Définitions STL = Standard Template Library

Il s'agit d'une bibliothèque générique qui fournit des solutions pour gérer un ensemble de données en utilisant des algorithmes efficaces

Du point de vue du programmeur, la STL fournit un groupe de classes répondant à divers besoins.

Tous les composants de la STL sont des templates.

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Composants

**Conteneurs**

Ils sont utilisés pour manipuler des objets d'un **même type**. On parle donc d'une **collection** d'objets  
La STL fournit différentes sortes de conteneurs pour combler différents besoins

**Itérateurs**

Les itérateurs sont des objets qui permettent de naviguer parmi les éléments d'un conteneur  
Un itérateur est un pointeur « intelligent »  
L'itérateur fait le lien entre les conteneurs et les algorithmes

**Algorithmes**

Les algorithmes de la STL offrent des services fondamentaux tels que recherche, tri, copie, modification des éléments des conteneurs.  
Les algorithmes sont des fonctions globales qui opèrent avec des itérateurs

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

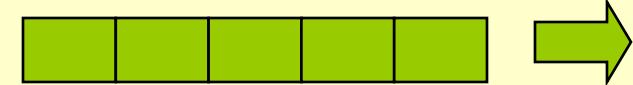
*Objet contenu*

*Itérateurs*

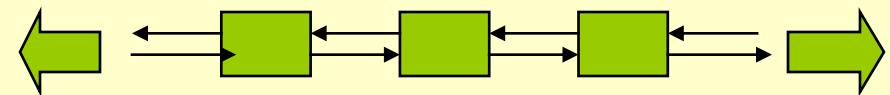
*Algorithmes*



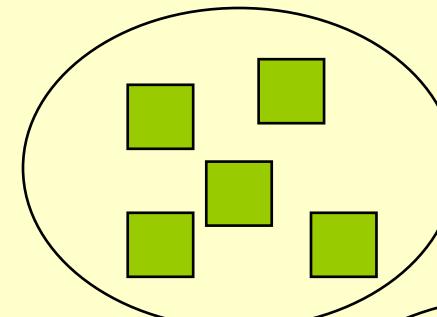
## Conteneurs



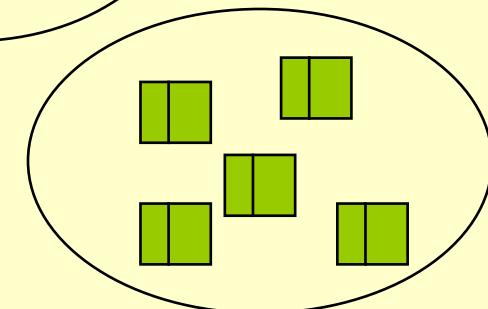
vector



list



set / multiset



map / multimap

## Conteneurs Associatifs

**STL**

*Introduction*

*Composants*

*Conteneurs*

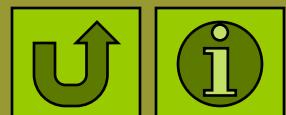
*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Conteneurs dérivés

- Le standard C++ fournit quelques conteneurs supplémentaires:
  - Piles (stacks): conteneur manipulant ses éléments selon la politique LIFO (last-in-first-out)
  - Files (queues): conteneur manipulant ses éléments selon la politique FIFO (first-in-first-out)
  - Queues de priorité: conteneur manipulant ses éléments selon la politique FIFO à priorité (un élément plus prioritaire se retrouve en tête de file)

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Choix du conteneur

- Les `vector` et les `deque` permettent d'accéder rapidement à un élément de la liste (opérateur `[]`) contrairement aux `list`
- Un élément peut être inséré
  - n'importe où pour une `list`
  - à la fin ou au début pour un `deque`
  - à la fin seulement pour un `vector`
- Utiliser un `vector` est un bon choix lorsque:
  - on a besoin d'insérer/supprimer des éléments seulement à la fin
  - le conteneur doit être compatible au tableau C standard



## Choix du conteneur (suite)

- Utiliser un `deque` est un bon choix lorsque:
  - on a besoin d'insérer/supprimer des éléments seulement à la fin ou au début
  - le conteneur ne doit pas être compatible avec un tableau C standard
  - la taille maximum requise du conteneur n'est pas connue
- Utiliser une `list` est un bon choix lorsque:
  - on a besoin d'insérer/supprimer des éléments au milieu du conteneur
  - le conteneur ne doit pas être compatible avec un tableau C standard
  - la taille maximum requise du conteneur n'est pas connue

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Objet contenu

L’élément qui sera inclus dans un conteneur doit posséder :

- Un constructeur copie
- Un opérateur d’affectation (operator =)
- Un destructeur
- Eventuellement, un constructeur par défaut
- Eventuellement, un test d’égalité (operator ==)
- Eventuellement, un critère de tri (operator <, ...)

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



# Opérations sur les itérateurs

**Opérateur \***

Retourne l'élément de la position courante

**Opérateur ++**

Fait pointer l'itérateur à l'élément suivant

**Opérateur ==**

Indique si 2 itérateurs pointent le même élément

**Opérateur =**

Assigne un itérateur

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



# Fonctions membres des conteneurs

Tous les conteneurs offrent les mêmes fonctions de base permettant aux itérateurs d'accéder aux éléments

- begin() : Retourne un itérateur représentant le début des éléments dans le conteneur
- end() : Retourne un itérateur représentant la fin des éléments dans le conteneur
- size() : Retourne le nombre d'éléments présents dans le conteneur

# STL

## Introduction

## Composants

## Conteneurs

## Conteneurs Dérivés

## Choix du conteneur

## Objet contenu

## Itérateurs

## Algorithmes



# Utilisation des itérateurs

- Tous les conteneurs définissent deux types d'itérateur
  - *container::iterator*  
→ permet de naviguer en mode lecture/écriture
  - *container::const\_iterator*  
→ permet de naviguer en mode lecture seulement

```
#include <list>
#include <iterator>
list<char> coll; // création d'une liste de caractères
for (char c = 'a'; c<='z'; c++)
    coll.push_back(c);
// impression de la liste à l'aide d'un itérateur
list<char>::const_iterator pos;
for(pos = coll.begin(); pos != coll.end(); pos++)
    cout << *pos << ' ';
```

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Exemples d'algorithmes

```
#include <vector>
vector<int> coll;
vector<int>::iterator pos;

// trouver le minimum et le maximum
pos = min_element(coll.begin(), coll.end());
cout << "min: " << *pos << endl;
pos = max_element(coll.begin(), coll.end());
cout << "max: " << *pos << endl;

// tri
sort(coll.begin(), coll.end());
```

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Plan Du Site

### Introduction

Composants

Conteneurs

Séquentiels

vector

list

deque

Associatifs

set

multiset

map

multimap

Conteneurs dérivés

Choix du conteneur

Objet contenu

Opérations sur les itérateurs

Fonctions membres des conteneurs

Utilisation des itérateurs

Exemples d'algorithmes

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Conteneurs Séquentiels

Les éléments du conteneur ont une position qui dépend du temps et de l'endroit de l'insertion

La position est donc indépendante de la valeur de l'élément

vector, list et deque sont des conteneurs séquentiels

# STL

*Introduction*

*Composants*

*Conteneurs*

*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Conteneurs Associatifs

- Les éléments du conteneur sont automatiquement triés lors de leur insertion selon un critère précis.
- Le critère de tri prend la forme d'une comparaison de la valeur de l'élément ou d'une clé spéciale définie pour l'élément.
- Les conteneurs associatifs sont généralement représentés sous forme d'arbre binaire
- set, multiset, map et multimap sont des conteneurs associatifs

# STL

## *Introduction*

## *Composants*

## *Conteneurs*

## *Conteneurs Dérivés*

## *Choix du conteneur*

## *Objet contenu*

## *Itérateurs*

## *Algorithmes*



## Vector

- Tableau dynamique, compatible avec les tableau du C standard.
- Les éléments sont insérés à la fin du tableau

```
#include <vector>
vector<int> coll;
```

```
for(int i=1; i<=6; i++)
    coll.push_back(i);
```

## Introduction

## Composants

## Conteneurs

## Conteneurs Dérivés

## Choix du conteneur

## Objet contenu

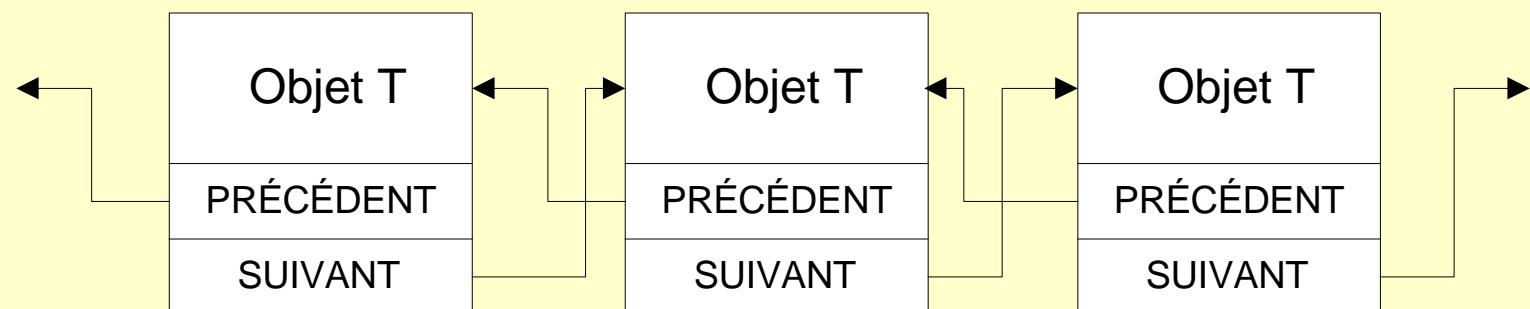
## Itérateurs

## Algorithmes



## List

- Liste doublement chaînée d’éléments
- Un élément pointe sur son prédecesseur et son successeur dans la liste
- Les éléments sont insérés n’importe où dans la liste



# STL

## *Introduction*

## *Composants*

## *Conteneurs*

## *Conteneurs Dérivés*

## *Choix du conteneur*

## *Objet contenu*

## *Itérateurs*

## *Algorithmes*



## Deque

- Deque = double-ended queue
- Tableau dynamique
- Les éléments sont insérés à la fin ou au début du tableau

```
#include <deque>
deque<int> coll;

for(int i=1; i<=6; i++) {
    coll.push_back(i);
    coll.push_front(i);
}
```

# STL

## Introduction

## Composants

## Conteneurs

## Conteneurs Dérivés

## Choix du conteneur

## Objet contenu

## Itérateurs

## Algorithmes

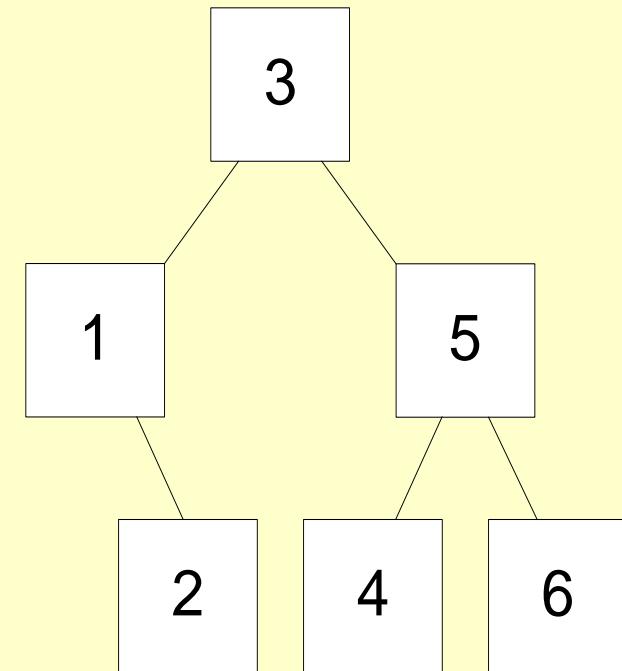


## Set

- Les éléments sont triés selon leur valeur. Chacun des éléments ne peut exister qu'une fois

```
#include <set>
set<int> coll;

coll.insert(3);
coll.insert(1);
coll.insert(5);
coll.insert(4);
coll.insert(1);
coll.insert(6);
coll.insert(2);
```



# STL

## Introduction

## Composants

## Conteneurs

## Conteneurs Dérivés

## Choix du conteneur

## Objet contenu

## Itérateurs

## Algorithmes

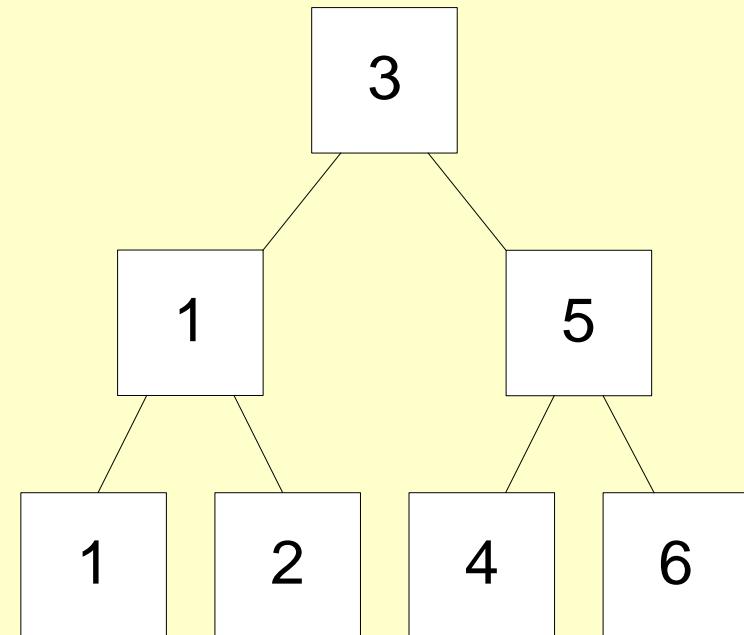


## Multiset

- Les éléments sont triés selon leur valeur. Chacun des éléments peut exister plus d'une fois

```
#include <set>
multiset<int> coll;

coll.insert(3);
coll.insert(1);
coll.insert(5);
coll.insert(4);
coll.insert(1);
coll.insert(6);
coll.insert(2);
```



# STL

## Map

### *Introduction*

### *Composants*

### *Conteneurs*

### *Conteneurs Dérivés*

### *Choix du conteneur*

### *Objet contenu*

### *Itérateurs*

### *Algorithmes*



- Les éléments sont une paire formée d'une clé jumelée à une valeur.
- Les éléments sont triés selon leur clé.
- Chacune des clés ne peut exister qu'une fois

```
#include <map>
map<string, float> coll;
```

```
coll["VAT"] = 0.15;    // VAT est la clé
coll["Pi"] = 3.1416;
coll["Un nombre"] = 4983.223;
coll["Null"] = 0;
```

# STL

*Introduction*

*Composants*

*Conteneurs*

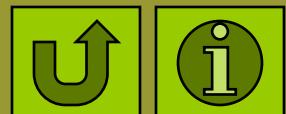
*Conteneurs Dérivés*

*Choix du conteneur*

*Objet contenu*

*Itérateurs*

*Algorithmes*



## Multimap

- Même chose que map excepté qu'une clé peut exister plus d'une fois.
- On peut donc retrouver des éléments possédant la même clé mais des valeurs différentes.
- Exemple : un dictionnaire