

Programmation Orientée Objet

Introduction à la STL (en C++)

et éléments sur les *templates*

Guillaume Revy

`guillaume.revy@univ-perp.fr`

Université de Perpignan **Via Domitia**



Plan du cours

1. Un transparent sur les *templates*

2. Standard Template Library (en C++)

- Introduction
- Les conteneurs
- Les itérateurs
- Quelques algorithmes

Plan du cours

1. Un transparent sur les *templates*

2. Standard Template Library (en C++)

- Introduction
- Les conteneurs
- Les itérateurs
- Quelques algorithmes

Qu'est ce que un *template* ?

↪ en un transparent...

- Un *template* ou *patron* ↪ modèle à partir duquel des classes pourront être générées automatiquement par le compilateur en fonction d'une série de paramètres : c'est le principe de *généricité*
 - ▶ *par exemple* : création d'une classe paramétrée par le type d'un de ses attributs

Qu'est ce que un *template* ?

↪ en un transparent...

- Un *template* ou *patron* ↪ modèle à partir duquel des classes pourront être générées automatiquement par le compilateur en fonction d'une série de paramètres : c'est le principe de **généricité**
 - ▶ **par exemple** : création d'une classe paramétrée par le type d'un de ses attributs
- **Exemple** : une classe **Intervalle**, paramétrée par le type de ces bornes

```
template <class T>    // la classe Intervalle sera parametree
                      // par le type T de ces elements
class Intervalle
{
    T binf, bsup;      // declaration de deux objets de type T

public:
    Intervalle() : binf(0), bsup(0) {}
    Intervalle(T binf, T bsup) : binf(binf), bsup(bsup) {}
    Intervalle(Intervalle const& other) : binf(other.binf), bsup(other.bsup) {}

    void print(void) { /* ... */ }
};
```

Qu'est ce que un *template* ?

↪ en un transparent...

- Un *template* ou *patron* ↪ modèle à partir duquel des classes pourront être générées automatiquement par le compilateur en fonction d'une série de paramètres : c'est le principe de **généricité**
 - ▶ **par exemple** : création d'une classe paramétrée par le type d'un de ses attributs
- **Exemple** : une classe **Intervalle**, paramétrée par le type de ces bornes

```
#include <iostream>

#include "Intervalle.hpp"

int
main(void)
{
    Intervalle< int > i1(0,1);
    Intervalle< int > i2;
    Intervalle< int > i3(i2);

    // ...
    return 0;
}
```

Plan du cours

1. Un transparent sur les *templates*

2. Standard Template Library (en C++)

- Introduction
- Les conteneurs
- Les itérateurs
- Quelques algorithmes

Qu'est ce que la STL ?

- La STL est une partie importante de la bibliothèque standard du C++

<http://www.sgi.com/tech/stl/>

- ▶ développée par Hewlett-Packard puis Silicon Graphics
 - ▶ 1994 : adoptée par le comité de standardisation du C++, pour être intégrée au langage
 - ▶ **attention** : l'amalgame est souvent fait entre STL et bibliothèque standard du C++
-
- Plus particulièrement, la bibliothèque standard du C++ est formée de trois grandes composantes :
 - ▶ la STL \rightsquigarrow **Standard Template Library**,
 - ▶ les autres outils de la bibliothèque standard du C++ : chaînes de caractères (`string`), flux d'entrée/sortie (`std::cin` et `std::cout`), ... ,
 - ▶ et les bibliothèques du langage C.

Qu'est ce que la STL ?

- La STL propose un grand nombre d'éléments \rightsquigarrow ensemble d'objets et de méthodes standards pour le C++
 - ▶ des structures de données évoluées (tableaux, listes chaînées, vecteurs, piles, ...),
 - ▶ des algorithmes efficaces sur ces structures de données (ajout, recherche, parcours, suppression, ...).

Qu'est ce que la STL ?

- La STL propose un grand nombre d'éléments \rightsquigarrow ensemble d'objets et de méthodes standards pour le C++
 - ▶ des structures de données évoluées (tableaux, listes chaînées, vecteurs, piles, ...),
 - ▶ des algorithmes efficaces sur ces structures de données (ajout, recherche, parcours, suppression, ...).

- Plus particulièrement, la STL propose principalement trois types d'éléments :
 - ▶ les **conteneurs** (containers) de base ou évolués (*adaptators*) \rightsquigarrow pour stocker les données (vecteur, listes, ...),
 - ▶ les **itérateurs** (iterators) \rightsquigarrow pour parcourir les conteneurs,
 - ▶ les algorithmes \rightsquigarrow travaillent sur les conteneurs via les itérateurs.

La STL en un transparent

- **Remarque** : la bibliothèque standard du C++ contient 51 fichiers d'entête, dont 13 appartiennent à la STL

```
#include <algorithm> // algorithmes utiles (tri, ...)
#include <deque>      // tableaux dynamiques (extensibles a chaque extremite)
#include <functional> // objets fonctions predefinis (unary_function, ...)
#include <iterator>   // iterateurs, fonctions, adaptateurs
#include <list>        // listes doublement chainee
#include <map>         // maps et multimaps
#include <numeric>     // fonctions numeriques : accumulation, ...
#include <queue>       // files
#include <set>         // ensembles
#include <stack>       // piles
#include <string>      // chaines de caracteres
#include <utility>     // divers utilitaires
#include <vector>      // vecteurs
```

Les conteneurs de la STL

- Les conteneurs sont des **structures de données abstraites**, permettant de contenir des données (\rightsquigarrow autres objets).
 - ▶ paramétrés par le type des éléments qu'ils contiennent
- Les principaux conteneurs de la STL sont :
 - ▶ les paires \rightsquigarrow `pair`
 - ▶ les vecteurs (\approx tableaux) \rightsquigarrow `vector`
 - ▶ les listes doublement chaînées \rightsquigarrow `list`
 - ▶ les tableaux dynamiques \rightsquigarrow `deque`
 - ▶ les ensembles ordonnés (au sens mathématique) \rightsquigarrow `set`
 - ▶ les tables associatives ordonnées (\approx tableaux pouvant être indexés par autre chose que des entiers, comme des chaînes de caractères) \rightsquigarrow `map`
 - ▶ les piles (structure LIFO) \rightsquigarrow `stack`
 - ▶ les files (structure FIFO) \rightsquigarrow `queue`

Les conteneurs de la STL

- Les conteneurs sont des **structures de données abstraites**, permettant de contenir des données (↪ autres objets).
 - ▶ paramétrés par le type des éléments qu'ils contiennent
- Les principaux conteneurs de la STL sont :
 - ▶ les paires ↪ `pair` conteneur simple
 - ▶ les vecteurs (≈ tableaux) ↪ `vector` conteneur séquentiel
 - ▶ les listes doublement chaînées ↪ `list` conteneur séquentiel
 - ▶ les tableaux dynamiques ↪ `deque` conteneur séquentiel
 - ▶ les ensembles ordonnés (au sens mathématique) ↪ `set` conteneur associatif
 - ▶ les tables associatives ordonnées (≈ tableaux pouvant être indexés par autre chose que des entiers, comme des chaînes de caractères) ↪ `map` conteneur associatif
 - ▶ les piles (structure LIFO) ↪ `stack` adaptateur de conteneur
 - ▶ les files (structure FIFO) ↪ `queue` adaptateur de conteneur

Les conteneurs de la STL

- Sur tous les conteneurs, un ensemble de méthodes est défini : `empty()`, `size()`, `clear()`, `erase(...)`, `begin()/end()`, `rbegin()/rend()`, opérateurs de comparaison, ...
- Différents types de conteneurs
 - ▶ conteneur simple (paire),
 - ▶ conteneurs séquentiels \rightsquigarrow stockent les éléments de manière séquentielle, et y accèdent de manière séquentielle ou directement (selon le conteneur)
 - ▶ conteneurs associatifs \rightsquigarrow associent une clé à chaque objet stocké, ces clés servant ensuite à accéder efficacement aux objets (éléments ordonnés)
 - ▶ adaptateurs de conteneurs \rightsquigarrow spécialisation de conteneurs (par exemple, `deque` spécialisé en `queue` ou `stack`)

Quelques complexités avant de commencer

	Insertion			Suppression			Accès			Recherche par valeur
	tête	qcq	queue	tête	qcq	queue	tête	qcq	queue	
vector	$O(n)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$			$O(n)$
deque	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	$O(1)$			$O(n)$
list	$O(1)$			$O(1)$			$O(1)$	$O(n)$	$O(1)$	$O(n)$
set/map	-	$O(\log n)$	-	-	$O(\log n)$	-	$O(1)$	-	$O(1)$	$O(\log n)$

Exemple de conteneurs

→ les vecteurs

```
#include <iostream>
#include <vector>    // <-- utilisation des vecteurs

using namespace std;

int
main(void)
{
    vector< int > v1;           // declaration d'un vecteur d'entiers vide v1
    vector< int > v2(8);       // declaration d'un vecteur de 8 cases non initialisees
    vector< int > v3(8,17);    // declaration d'un vecteur de 8 cases initialisees a 17

    cout << "Taille de v1 : " << v1.size() << endl;    // --> Taille de v1 : 0
    cout << "Taille de v2 : " << v2.size() << endl;    // --> Taille de v2 : 8
    cout << "Taille de v3 : " << v2.size() << endl;    // --> Taille de v3 : 8

    for(int i = 0 ; i < v3.size() ; i++){
        cout << "v3.at(" << i << ") : " << v3.at(i); // indexation via la methode at()
        cout << " ou v3[" << i << "] : " << v3[i] << endl; // indexation via les []
    }

    return 0;
}
```

- **Attention** : le parcours, simple avec les vecteurs via l'opérateur [], ne l'est pas pour les autres conteneurs → passage par les itérateurs.

Exemple de conteneurs

↪ une matrice avec des vecteurs

```
#include <iostream>
#include <vector>    // <-- utilisation des vecteurs

using namespace std;

typedef vector< float > Line;
typedef vector< Line > Matrix;

int
main(void)
{
    Line l(17,0.0);           // declaration d'une ligne de 17 elements nuls
    Matrix m(17,l);           // declaration d'une matrice de 17 lignes

    // vector< vector< float > > m;
    // --> necessite d'initialiser m "a la main"

    for(int i = 0 ; i < m.size() ; i++){
        for(int j = 0 ; j < m[i].size() ; j++){
            cout << m[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Exemple de conteneurs

↪ les listes doublement chaînées

```
#include <iostream>
#include <list>      // <-- utilisation des listes doublement chainees

using namespace std;

int
main(void)
{
    list< int > L;      // declaration similaire a celle d'un vecteur
                        // mais utilisation differente

    L.push_back(5);     // L = {5}
    L.push_back(4);     // L = {5,4}
    L.push_front(3);    // L = {3,5,4}

    cout << "Tete de L : " << L.front() << endl; // Tete de L : 3
    cout << "Queue de L : " << L.back() << endl; // Queue de L : 4

    for(int i = 0 ; i < L.size() ; i++){
        cout << "L.at(" << i << ") : " << L.at(i);    // ERREUR
        cout << "L[" << i << "] : " << L[i] << endl;  // ERREUR
    }

    return 0;
}
```

Exemple de conteneurs

↪ les listes doublement chaînées

```
int
main(void)
{
    list< int > L;           // declaration similaire a celle d'un vecteur
                             // mais utilisation differente

    L.push_back(5);         // L = {5}
    L.push_back(4);         // L = {5,4}
    L.push_front(3);        // L = {3,5,4}

    cout << "Tete de L : " << L.front() << endl; // Tete de L : 3
    cout << "Queue de L : " << L.back() << endl; // Queue de L : 4

    for(int i = 0 ; i < L.size() ; i++){
        cout << "L.at(" << i << ") : " << L.at(i);    // ERREUR
        cout << "L[" << i << "] : " << L[i] << endl;  // ERREUR
    }

    return 0;
}
```

■ Remarques

- ▶ la méthode `push_back` est également définie sur les vecteurs
- ▶ par contre, pour les listes, l'indexation via l'opérateur `[]` ou la méthode `at(...)` n'est pas autorisée

Exemple de conteneurs

↪ utilisation d'un conteneur associatif

```
#include <iostream>
#include <map>           // <-- utilisation des tables associatifs (map)

using namespace std;

int
main(void)
{
    map< string, string > telephones; // tableau indexe par une
                                      // chaine de caracteres

    telephones["Nom1"] = "0132655698";
    telephones["Nom2"] = "0686423657";

    cout << "Nom1 -> " << telephones["Nom1"] << endl;
    cout << "Nom2 -> " << telephones["Nom2"] << endl;

    return 0;
}
```

Exemple de conteneurs

↪ utilisation d'un conteneur associatif

```
#include <iostream>
#include <map>           // <-- utilisation des tables associatifs (map)

using namespace std;

int
main(void)
{
    map< string, string > telephones; // tableau indexe par une
                                      // chaine de caracteres

    telephones["Nom1"] = "0132655698";
    telephones["Nom2"] = "0686423657";

    cout << "Nom1 -> " << telephones["Nom1"] << endl;
    cout << "Nom2 -> " << telephones["Nom2"] << endl;

    return 0;
}
```

```
Nom1 -> 0132655698
Nom2 -> 0686423657
```

Qu'est ce qu'un itérateur ?

- Les itérateurs sont des objets fondamentaux de la STL \rightsquigarrow ils permettent de manipuler les conteneurs de manière transparente, plus facilement
- Un itérateur permet :
 - ▶ de parcourir un conteneur,
 - ▶ d'accéder aux données contenues dans le conteneur,
 - ▶ et (éventuellement) de modifier les données.
- Un itérateur peut être vu comme un pointeur sur un élément du conteneur.
- Il existe deux types d'itérateurs :
 - ▶ `iterator` ou `const_iterator` \rightsquigarrow parcours d'un conteneur du début à la fin
 - ▶ `reverse_iterator` ou `const_reverse_iterator` \rightsquigarrow parcours d'un conteneur en sens inverse (de la fin au début)

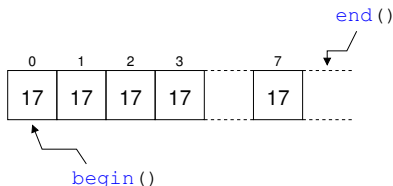
Utilisation générale d'un itérateur

- Un itérateur associé à un conteneur `C` (vecteur, liste, ...) se déclare de la manière suivante.

```
C::iterator it1;  
C::reverse_iterator it2;
```

- Un itérateur `iterator` peut être initialisé grâce aux méthodes :
 - ▶ `begin()` \rightsquigarrow retourne un itérateur "pointant" sur le premier élément du conteneur
 - ▶ `end()` \rightsquigarrow retourne un itérateur "pointant" sur l'élément juste après le dernier élément du conteneur

ou toutes autres méthodes retournant un itérateur sur le conteneur : `find()`, ...



Utilisation générale d'un itérateur

- Un itérateur associé à un conteneur `C` (vecteur, liste, ...) se déclare de la manière suivante.

```
C::iterator it1;  
C::reverse_iterator it2;
```

- Un itérateur `iterator` peut être initialisé grâce aux méthodes :
 - ▶ `begin()` \rightsquigarrow retourne un itérateur "pointant" sur le premier élément du conteneur
 - ▶ `end()` \rightsquigarrow retourne un itérateur "pointant" sur l'élément juste après le dernier élément du conteneurou toutes autres méthodes retournant un itérateur sur le conteneur : `find()`, ...
- De manière similaire, un itérateur `reverse_iterator` peut être initialisé en utilisant les méthodes : `rbegin()`/`rend()`, ...
- Accéder à l'élément "pointé" par un itérateur `it` se fait de la même manière que pour accéder à l'élément pointé par un pointeur \rightsquigarrow `*it`

Exemple d'utilisation d'un itérateur

↪ pour le parcours d'un vecteur

```
#include <iostream>
#include <vector>    // <-- utilisation des vecteurs

using namespace std;

int
main(void)
{
    vector< int > v1;           // declaration d'un vecteur d'entiers vide v1
    vector< int > v2(8);       // declaration d'un vecteur de 8 cases non initialisees
    vector< int > v3(8,17);    // declaration d'un vecteur de 8 cases initialisees a 17

    cout << "Taille de v1 : " << v1.size() << endl; // --> Taille de v1 : 0
    cout << "Taille de v2 : " << v2.size() << endl; // --> Taille de v2 : 8
    cout << "Taille de v3 : " << v2.size() << endl; // --> Taille de v3 : 8

    for(int i = 0 ; i < v3.size() ; i++){
        cout << "v3.at(" << i << ") : " << v3.at(i); // indexation via la methode at()
        cout << " ou v3[" << i << "] : " << v3[i] << endl; // indexation via les []
    }

    return 0;
}
```

Exemple d'utilisation d'un itérateur

↪ pour le parcours d'un vecteur

```
#include <iostream>
#include <vector>    // <-- utilisation des vecteurs

using namespace std;

int
main(void)
{
    vector< int > v1(8);    // declaration d'un vecteur de 8 cases non initialisees
    vector< int >::iterator it1;
    vector< int >::reverse_iterator it2;
    int idx = 0;

    // --> initialisation du debut a la fin du vecteur
    for(it1 = v1.begin() ; it1 != v1.end() ; it1++)
        (*it1) = (idx++);    // affectation de (idx++) a l'element pointe par it1,
                             // c'est-a-dire, a v1[idx]

    // --> parcours dans le sens inverse
    for(it2 = v1.rbegin() ; it2 != v1.rend() ; it2++)
        cout << "v1[" << idx-- << "] = " << (*it2) << endl;    // affichage de l'element
                                                                    // pointe par it2

    return 0;
}
```

Exemple d'utilisation d'un itérateur

↪ affichage de la matrice

```
#include <iostream>
#include <vector>    // <-- utilisation des vecteurs

using namespace std;

typedef vector< float > Line;
typedef vector< Line > Matrix;

int
main(void)
{
    Line l(17,0.0);           // declaration d'une ligne de 17 elements nuls
    Matrix m(17,l);           // declaration d'une matrice de 17 lignes

    Matrix::iterator i;
    Line::iterator j;

    for(i = m.begin() ; i != m.end() ; i++){
        for(j = (*i).begin() ; j != (*i).end() ; j++){
            cout << (*j) << " ";
            cout << endl;
        }
    }

    return 0;
}
```

Exemple d'utilisation d'un itérateur

↪ les listes doublement chaînées

```
#include <iostream>
#include <list>      // <-- utilisation des listes doublement chainees

using namespace std;

int
main(void)
{
    list< int > L;      // declaration similaire a celle d'un vecteur
                       // mais utilisation differente

    L.push_back(5);     // L = {5}
    L.push_back(4);     // L = {5,4}
    L.push_front(3);    // L = {3,5,4}

    cout << "Tete de L : " << L.front() << endl; // Tete de L : 3
    cout << "Queue de L : " << L.back() << endl; // Queue de L : 4

    for(int i = 0 ; i < L.size() ; i++){
        cout << "L.at(" << i << ") : " << L.at(i);    // ERREUR
        cout << "L[" << i << "] : " << L[i] << endl;    // ERREUR
    }

    return 0;
}
```

Exemple d'utilisation d'un itérateur

↪ les listes doublement chaînées

```
int
main(void)
{
    list< int > L;           // declaration similaire a celle d'un vecteur
                             // mais utilisation differente

    L.push_back(5);          // L = {5}
    L.push_back(4);          // L = {5,4}
    L.push_front(3);         // L = {3,5,4}

    cout << "Tete de L : " << L.front() << endl; // Tete de L : 3
    cout << "Queue de L : " << L.back() << endl; // Queue de L : 4

    int idx = 0;
    list< int >::iterator it1;
    list< int >::reverse_iterator it2;
    for(it1 = L.begin() ; it1 != L.end() ; it1++)
        cout << "L(" << (idx++) << " ) : " << (*it1) << endl;    // OK
    // --> 3, 5, 4

    for(it2 = L.rbegin() ; it2 != L.rend() ; it2++)
        cout << "L(" << (idx++) << " ) : " << (*it2) << endl;    // OK
    // --> 4, 5, 3

    return 0;
}
```

Exemple d'utilisation d'un itérateur

↪ les tableaux associatifs (map)

```
#include <iostream>
#include <map>          // <-- utilisation des tables associatifs (map)

using namespace std;

int
main(void)
{
    map< string, string > telephones; // tableau indexé par une
                                      // chaîne de caractères

    telephones["Nom1"] = "0132655698";
    telephones["Nom2"] = "0686423657";
    telephones["Nom3"] = "0682749823";
    telephones["Nom4"] = "0221331090";

    map< string, string >::iterator it;
    for(it = telephones.begin() ; it != telephones.end() ; it++)
        // it pointe sur une paire :
        // - premier élément : it->first ou (*it).first
        // - second élément : it->second ou (*it).second
        cout << it->first << " -> " << (*it).second << endl;

    return 0;
}
```

Exemple d'utilisation d'un itérateur

↪ recherche et suppression dans un tableau associatif (map)

```
// --> recherche du numero de "Nom1"
string nom("Nom5");
it = telephones.find(nom);
if(it == telephones.end())
    cout << "Error : \" << nom << "\" not found..." << endl;
else
    cout << "Error : \" << nom << "\" found..." << endl;

nom = "Nom1";
it = telephones.find(nom);
if(it == telephones.end())
    cout << "Error : \" << nom << "\" not found..." << endl;
else
    cout << "Error : \" << nom << "\" not..." << endl;

// --> suppression de l'entree indexee par "Nom1"
telephones.erase(it);

for(it = telephones.begin() ; it != telephones.end() ; it++)
    cout << it->first << " -> " << (*it).second << endl;
```

```
Error : "Nom5" not found...
Error : "Nom1" not...
Nom2 -> 0686423657
Nom3 -> 0682749823
Nom4 -> 0221331090
```

Les algorithmes de la STL

- Finalement, la STL offre un ensemble d'algorithmes utilisables sur les conteneurs, via les itérateurs.

- Les principaux algorithmes sont :
 - ▶ les algorithmes numériques \rightsquigarrow minimum, maximum, sommes partielles, accumulations, produits, ...),
 - ▶ les algorithmes de tri \rightsquigarrow quick-sort, ...,
 - ▶ les algorithmes modifiant les conteneurs \rightsquigarrow remplissage, copie, échanges, union, intersection, ...,
 - ▶ les algorithmes ne modifiant pas les conteneurs \rightsquigarrow recherche, applications de fonctions, inclusion, ...

Exemple d'utilisation d'algorithme de la STL

↪ sommer les éléments d'un vecteur d'entiers

```
#include <iostream>
#include <vector>      // <-- utilisation des vecteurs
#include <numeric>     // <-- utilisation des algorithmes numeriques

using namespace std;

int
main(void)
{
    vector< int > v1(8);    // declaration d'un vecteur de 8 cases non initialisees
    vector< int >::iterator it1;
    int idx = 1;

    // --> initialisation du debut a la fin du vecteur
    for(it1 = v1.begin() ; it1 != v1.end() ; it1++)
        (*it1) = (idx++);

    // --> calcule de la somme des elements : 1 + 2 + ... + 8
    int sum = accumulate(v1.begin(), v1.end(), 0);
    cout << "Sum : " << sum << endl;

    return 0;
}
```

Exemple d'utilisation d'algorithme de la STL

→ trier les éléments d'un vecteur d'entiers

```
//...
#include <algorithm> // <-- utilisation des algorithmes de tri

int
main(void)
{
    vector< int > v1(8); // declaration d'un vecteur de 8 cases non initialisees
    vector< int >::iterator it1;

    // --> initialisation du debut a la fin du vecteur
    for(it1 = v1.begin() ; it1 != v1.end() ; it1++) (*it1) = rand()%101;

    // --> affichage des elements du vecteur
    for(it1 = v1.begin() ; it1 != v1.end(); it1++)
        cout << (*it1) << " ";
    cout << endl; // 32 32 54 12 52 56 8 30

    // --> tri des elements du vecteur
    sort(v1.begin(), v1.end());

    // --> affichage des elements du vecteur trie
    for(it1 = v1.begin() ; it1 != v1.end(); it1++)
        cout << (*it1) << " ";
    cout << endl; // 8 12 30 32 32 52 54 56

    return 0;
}
```

Exemple d'utilisation d'algorithme de la STL

↪ faire l'union des éléments de deux vecteurs d'entiers

```
//...
#include <algorithm>    // <-- utilisation de l'union
#include <iterator>     // <-- ostream_iterator

int
main(void)
{
    vector< int > v1(4), v2(4);    // declaration de deux vecteurs de 4 cases non
                                   // initialisees

    vector< int >::iterator it1;

    // --> initialisation des deux vecteurs
    for(it1 = v1.begin() ; it1 != v1.end() ; it1++) (*it1) = rand()%101;
    for(it1 = v2.begin() ; it1 != v2.end() ; it1++) (*it1) = rand()%101;

    // --> affichage des elements des vecteurs
    for(it1 = v1.begin() ; it1 != v1.end(); it1++)
        cout << (*it1) << " ";
    cout << endl;                                     // 32 32 54 12
    for(it1 = v2.begin() ; it1 != v2.end(); it1++)
        cout << (*it1) << " ";
    cout << endl;                                     // 52 56 8 30

    // --> affichage de l'union des deux vecteurs
    set_union(v1.begin(), v1.end(), v2.begin(), v2.end(),
              ostream_iterator<int>(cout, " "));
                                                    // 32 32 52 54 12 56 8 30

    return 0;
}
```

Exemple d'utilisation d'algorithme de la STL

↪ recherche d'un élément dans un vecteur

```
//...
#include <algorithm> // <-- utilisation des algorithmes de recherche

int
main(void)
{
    vector< int > v1(8); // declaration d'un vecteur de 8 cases non initialisees
    vector< int >::iterator it1;
    int idx = 0;

    // --> initialisation du debut a la fin du vecteur
    for(it1 = v1.begin() ; it1 != v1.end() ; it1++)
        (*it1) = (idx++);

    vector< int >::iterator it;

    // --> recherche du numero 7
    it = find(v1.begin(), v1.end(), 7);
    if(it == v1.end())
        cout << "Error : 7 not found..." << endl;
    else
        cout << "Error : 7 found..." << endl;
    // Error : 7 found...

    return 0;
}
```

Ce cours est une brève introduction à la STL, et de son efficacité... Je vous conseille d'aller voir par vous même à l'adresse suivante pour en savoir plus !

`http://www.sgi.com/tech/stl/`

Questions ?