**La Librairie Standard De C++   
(STL)**

La librairie standard du C++ est née de la volonté d'apporter aux programmeurs C++ un canevas de programmation efficace, générique et simple à utiliser. Celui-ci comprend notamment :

* Des classes conteneur génériques et des algorithmes associés
* Une classe **string** paramétrée par le type de caractères qu'elle contient
* Une nouvelle famille de classes flux (**iostream**).

### Intérêt d’utiliser la STL :

* **Fiabilité** : les classes de la STL sont particulièrement sûres, et, dès qu'une erreur est découverte, elle est rapidement corrigée1 En outre, la plupart des composants a été écrite par des spécialistes.
* **Réutilisabilité** : la STL étant dorénavant totalement intégrée au standard du C++, son utilisation est sensée garantir la portabilité du logiciel.
* **Compréhensibilité et maintenabilité** : la standardisation de la STL garantit que tout utilisateur sera capable d'appréhender favorablement du code reposant sur elle.
* **Efficacité** : les composants ayant été écrits par des maîtres, il y a fort à parier que leur utilisation conduit à du code plus efficace que celui reposant sur des bibliothèques « maison »
* Présenter de manière cohérente la STL est assez difficile. En effet, il est impossible de séparer l'étude des conteneurs de celle des itérateurs ou des algorithmes tant ces notions sont mêlées. Aussi, cela contraindra le lecteur à des aller retours fréquents dans le texte.

## Les strings

La classe [std::string](http://www.dinkumware.com/htm_cpl/string2.html) permet de gérer des chaînes de caractères.

#include <string>

using namespace std;

int main ()

{

string text = "abc defg";

string words[] = { "aaa", "bbb", "ccc", "ddd", "eee" };

text += words[0];

return (0);

}

### Les principales opérations avec les strings

* Assignation des strings

string s = "abcd";

string t = s;

t += "abcd";

* Longueur d'un string

string s = "abcd";

int size = s.size ();

* Recherche dans un string

string s = "abcd";

int pos = s.find ("bc");

* Récupération du contenu pour passer aux fonctions C

string s = "abcd";

if (!strcmp (s.c\_str (), "abcd")) ...

...

* Utilisation avec la fonction std::getline

ifstream f; // declaration d’un fichier

string s;

getline (f, s);

...

## Les vecteurs

La classe [std::vector<T>](http://www.dinkumware.com/htm_cpl/vector.html) permet de mémoriser tout type d'objet séquentiellement.

* des types de base:

#include <vector>

Using namespace std ;

int main ()

{

vector<int> v;

for (int i = 0; i < 10; i++)

{

v.push\_back (i); //ajout d’un element au vecteur à la fin

}

return (0);

}

* des objets utilisateurs:

#include <vector>

class A

{

public:

A (int value) { \_value = value; } //constructeur

private:

int \_value;

};

int main ()

{

std::vector<A> v;

for (int i = 0; i < 10; i++)

{

v.push\_back (A(i));

}

return (0);

}

* des *références* ou des *pointeurs* sur des objets utilisateurs :

#include <vector>

class A

{

public:

A (int value) { \_value = value; }

private:

int \_value;

};

int main ()

{

std::vector<A\*> v;

for (int i = 0; i < 10; i++)

{

v.push\_back (new A(i));

}

return (0);

}

### Les principales opérations possibles sur un vecteur

* Ajout d'un élément à la fin d'un vecteur

v.push\_back (12);

* Suppression du dernier élément d'un vecteur

v.pop\_back (); // enleve un element

* Suppression de tous les éléments d'un vecteur

v.clear ();

* Accès au premier et au dernier élément d'un vecteur

int first = v.front (); //premier element

int last = v.back (); //dernier élément

* Accès à un élément d'un vecteur par sa position

int i = v[2];

### Les itérations dans un vecteur

On peut itérer sur les éléments d'un vecteur en utilisant un des itérateurs prédéfinis:

* std::vector<T>::iterator //iterateur classique
* std::vector<T>::const\_iterator //iterateur avec acces en lecture
* std::vector<T>::reverse\_iterator
* std::vector<T>::const\_reverse\_iterator

#include <iostream>

#include <vector>

typedef std::vector<int> int\_vector; // définition d'un type vecteur d'entiers

using namespace std;

int main ()

{

int\_vector v; // v est un veteur d’entier

// creation d'un tableau de 10 entiers

for (int i = 0; i < 10; i++)

{

v.push\_back (i); //on ajoute un élément à la suite du dernier

}

int\_vector::iterator **it**; // déclaration d'un iterateur sur un tableau d'entier

for (it = v.begin (); it != v.end (); ++it)

{

int i = \*it; // i=on stocke dans i le contenu de l'itérateur

std::cout << "i = " << i << std::endl;

}

return (0);

}

### Les opérations avec itérateurs sur un vecteur

* Suppression d'un élément d'un vecteur

vector<int> v;

vector<int>::iterator it;

it=v.begin()+7; // on place l’iterateur à 7 élément après le début

v.erase (it); // on efface l’element pointé par it

* Insertion d'un élément dans un vecteur

vector<int> v;

vector<int>::iterator it;

it = v.begin ();

// Insertion avant l'élément pointé par it

v.insert (it, 24); // on ajoute la valeur 24 avant l’élement pointé par it

## Les listes

Les listes fournies par la classe [list<T>](http://www.dinkumware.com/htm_cpl/list.html" \t "_blank) fonctionnent de façon très similaire aux vecteurs, mais proposent une implémentation à base de listes chaînées bidirectionnelles permettant des insertions et suppressions plus performantes.

#include <list>

#include <iostream>

using namespace std;

typedef list<int> int\_list; // definition d'un alias pour une liste chainé d'entiers

int main ()

{

int\_list liste; // liste est de type liste chainée d'entiers

for (int i = 0; i < 10; i++)

{

liste.push\_front (i); //insertion en tete

liste.push\_back (i); //insertion en queue

}

int\_list::iterator it; // déclaration d'un itérateur int\_list d'entiers

// affichage

for (it = liste.begin (); it != liste.end (); ++it)

{

int i = \*it;

cout << "i = " << i << endl;

}

}

## Les *maps*

La classe [map](http://www.dinkumware.com/htm_cpl/map.html" \t "_blank) fournit une collection de paires clé/valeur. La collection est ordonnée automatiquement d'auprès un critère utilisé pour la clé actuelle.

#include <iostream>

#include <map>

#include <string>

typedef std::map<std::string, int> annuaire;

int main ()

{

annuaire a;

a["Perus"] = 164468443;

a["Arnault"] = 164468424;

a["Jouvin"] = 164468932;

a["Lefebvre"] = 169155240;

a["Garnier"] = 164468551;

a["Hrivnacova"] = 169156594;

annuaire::iterator it;

for (it = a.begin (); it != a.end (); ++it)

{

std::string key = (\*it).first;

int number = (\*it).second;

std::cout << "name = " << key << " number = " << number << std::endl;

}

}

## Quelques références

1. [SGI: Standard Template Library Programmer's Guide](http://www.sgi.com/tech/stl/)

(http://www.sgi.com/tech/stl/)

1. [STL Overview (from Rob Kremer - Calgary)](http://pages.cpsc.ucalgary.ca/~kremer/STL/monitor/index.html)

(http://pages.cpsc.ucalgary.ca/~kremer/STL/monitor/index.html)