Cours de C++

Les templates et la STL

Gauthier Quesnel
quesnel@lil.univ-littoral.fr

Laboratoire d'Informatique du Littoral

3 avril 2006

Plan

Les templates

- La philosophie du **couper/remplacer** :

Les templates

- La philosophie du **couper/remplacer** :

Example

```
template <class C >                                          1
std::string to_string(const C& c)                            2
{                                                            3
    std::ostringstream o;                                    4
    o << c;                                                  5
    return o.str();                                          6
}                                                            7
                                                             8
std::string to_string < double >(3.1233);                    9
std::string to_string < int >(3);                            10
std::string to_string < bool >(true);                        11
std::string to_string(2.12308);                              12
```

Les templates

- Remplace le template par la donnée :

Les templates

- Remplace le template par la donnée :

Example

```
std::string to_string < double >(3.1233);

// Est équivalent à :

std::string to_string(const double& c)
{
    std::ostringstream o;
    o << c;
    return o.str();
}
```

Plan

Les templates

- Les templates : **programmation générique** en français.

Les templates

- Les templates : **programmation générique** en français.
- Les types fournis au template peuvent être des types simples, des classes ou même des fonctions.

Les templates

- Les templates : **programmation générique** en français.
- Les types fournis au template peuvent être des types simples, des classes ou même des fonctions.

Example

```
std::vector < int > tab(100, 0);                                    1
std::generate(tab.begin(), tab.end(), rand);                        2
                                                                    3
// la fonction dans l'API C++ :                                     4
template < typename _ForwardIterator, typename _Generator>          5
void std::generate(_ForwardIterator __first,                        6
                   _ForwardIterator __last,                         7
                   _Generator __gen)                                8
{                                                                   9
    for (; __first != __last; ++__first)                            10
        *__first = __gen();                                         11
}                                                                   12
```

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :
  - des structures de données, tableaux, listes chaînes, piles.

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :
  - des structures de données, tableaux, listes chaînes, piles.
  - des algorithmes de recherches, suppression, de tas.

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :
  - des structures de données, tableaux, listes chaînes, piles.
  - des algorithmes de recherches, suppression, de tas.
- Il existe principalement trois types d'éléments :

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :
  - des structures de données, tableaux, listes chaînes, piles.
  - des algorithmes de recherches, suppression, de tas.
- Il existe principalement trois types d'éléments :
  - les **conteneurs** : stockent les données.

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :
  - des structures de données, tableaux, listes chaînes, piles.
  - des algorithmes de recherches, suppression, de tas.
- Il existe principalement trois types d'éléments :
  - les **conteneurs** : stockent les données.
  - les **itérateurs** : parcourent les conteneurs.

Standard Template Library

- Une partie **très importante** du langage C++ : la STL (Standard Template Library)
- La STL propose un grand nombre d'éléments :
  - des structures de données, tableaux, listes chaînes, piles.
  - des algorithmes de recherches, suppression, de tas.
- Il existe principalement trois types d'éléments :
  - les **conteneurs** : stockent les données.
  - les **itérateurs** : parcourent les conteneurs.
  - les **algorithmes** : travaillent sur les conteneurs à partir des itérateurs.

Les conteneurs

- Les conteneurs permettent de contenir des éléments : la généricité paramétrera le type des éléments à utiliser.

Les conteneurs

- Les conteneurs permettent de contenir des éléments : la généricité paramétrera le type des éléments à utiliser.
- Plusieurs types de conteneurs sont utilisables : les tableaux, classes vector et deque, les listes list, les piles stack, les queue, les ensembles set, les ensembles multiples multiset, les cartes map et les multicartes multimap.

Les conteneurs

- Les conteneurs permettent de contenir des éléments : la généricité paramétrera le type des éléments à utiliser.
- Plusieurs types de conteneurs sont utilisables : les tableaux, classes vector et deque, les listes list, les piles stack, les queue, les ensembles set, les ensembles multiples multiset, les cartes map et les multicartes multimap.
- Il n'y a pas de **classe générique de base** (classe Object du java), par exemple). Cette option aurait pu être intéressante sur bien des aspects, mais il ne faut jamais oublier que l'un des buts fondamentaux de la librairie STL est d'être la plus rapide possible (et la liaison dynamique à un coup).

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :
  - Accès aléatoires : tableaux.

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :
  - Accès aléatoires : tableaux.
  - Accès linéaires : les listes, les maps, les tableaux.

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :
  - Accès aléatoires : tableaux.
  - Accès linéaires : les listes, les maps, les tableaux.
  - etc.

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :
  - Accès aléatoires : tableaux.
  - Accès linéaires : les listes, les maps, les tableaux.
  - etc.
- Ils permettent de manipuler les conteneurs de manière transparente.

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :
  - Accès aléatoires : tableaux.
  - Accès linéaires : les listes, les maps, les tableaux.
  - etc.
- Ils permettent de manipuler les conteneurs de manière transparente.
- Un itérateur peut-être vu comme un pointeur sur un élément d'un conteneur.

Les itérateurs

- Les itérateurs sont **fondamentaux** dans la STL !
- Ils en existent de plusieurs types :
  - Accès aléatoires : tableaux.
  - Accès linéaires : les listes, les maps, les tableaux.
  - etc.
- Ils permettent de manipuler les conteneurs de manière transparente.
- Un itérateur peut-être vu comme un pointeur sur un élément d'un conteneur.
- Par définition, un itérateur permet de récupérer une donnée (pour la manipuler) et de passer à la suivante pour mettre en place l'itération.

Algorithmes et allocateurs

- Les algorithmes :

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.
  - Les algorithmes utilisent des itérateurs linéaire ou à accès aléatoire → tous les conteneurs peuvent appliquer les algorithmes.

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.
  - Les algorithmes utilisent des itérateurs linéaire ou à accès aléatoire → tous les conteneurs peuvent appliquer les algorithmes.
  - Il en existe beaucoup :

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.
  - Les algorithmes utilisent des itérateurs linéaire ou à accès aléatoire → tous les conteneurs peuvent appliquer les algorithmes.
  - Il en existe beaucoup :
    - std::find : cherche quelque chose

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.
  - Les algorithmes utilisent des itérateurs linéaire ou à accès aléatoire → tous les conteneurs peuvent appliquer les algorithmes.
  - Il en existe beaucoup :
    - std::find : cherche quelque chose
    - std::fill : rempli un conteneur avec une donnée.

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.
  - Les algorithmes utilisent des itérateurs linéaire ou à accès aléatoire → tous les conteneurs peuvent appliquer les algorithmes.
  - Il en existe beaucoup :
    - std::find : cherche quelque chose
    - std::fill : rempli un conteneur avec une donnée.
    - std::generate : génére des données à partir d'une fonction.

Algorithmes et allocateurs

- Les algorithmes :
  - Les algorithmes permettent de manipuler les données d'un conteneur de manière transparente.
  - Les algorithmes utilisent des itérateurs linéaire ou à accès aléatoire → tous les conteneurs peuvent appliquer les algorithmes.
  - Il en existe beaucoup :
    - std::find : cherche quelque chose
    - std::fill : rempli un conteneur avec une donnée.
    - std::generate : génére des données à partir d'une fonction.
- Les allocateurs permettent de choisir entre plusieurs possibilités d'allocation de mémoire pour les conteneurs.

Plan

Tableau de caracatères : char

- Les chaînes de caractères de type C sont compatibles avec le langage C++. Certaines classes C++ utilisent des char[].

Tableau de caracatères : char

- Les chaînes de caractères de type C sont compatibles avec le langage C++. Certaines classes C++ utilisent des char[].

Example

```
#include <fstream>                                    1
                                                      2
int main(int argc, char** argv)                       3
{                                                     4
    int i = 0;                                        5
                                                      6
    if (argc == 2) {                                  7
        std::ifstream o(argv[1]);                     8
        o >> i;                                       9
    }                                                 10
    std::cout << i;                                   11
}                                                     12
```

std::string

- La classe std::string encapsule un buffer de caractères.

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».
- Une API riche :

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».
- Une API riche :
  - operator[](size_t) : l'accès à un char.

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».
- Une API riche :
  - operator[](size_t) : l'accès à un char.
  - c_str() et data() : la conversion de la chaîne de caractères en tableau de caractères avec ou sans le caractère fin de ligne.

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».
- Une API riche :
  - operator[](size_t) : l'accès à un char.
  - c_str() et data() : la conversion de la chaîne de caractères en tableau de caractères avec ou sans le caractère fin de ligne.
  - append, operator+, operator+=, insert, replace, erase, assign, operator= : les fonctions d'ajout, d'insertion, de suppression, d'affectation.

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».
- Une API riche :
  - operator[](size_t) : l'accès à un char.
  - c_str() et data() : la conversion de la chaîne de caractères en tableau de caractères avec ou sans le caractère fin de ligne.
  - append, operator+, operator+=, insert, replace, erase, assign, operator= : les fonctions d'ajout, d'insertion, de suppression, d'affectation.
  - find, rfind, find_first_of, find_first_not_of, find_last_of : les fonctions de recherche.

std::string

- La classe std::string encapsule un buffer de caractères.
- Elle peut contenir plusieurs caractères de fin de ligne « 0 ».
- Une API riche :
  - operator[](size_t) : l'accès à un char.
  - c_str() et data() : la conversion de la chaîne de caractères en tableau de caractères avec ou sans le caractère fin de ligne.
  - append, operator+, operator+=, insert, replace, erase, assign, operator= : les fonctions d'ajout, d'insertion, de suppression, d'affectation.
  - find, rfind, find_first_of, find_first_not_of, find_last_of : les fonctions de recherche.
- `http://www.sgi.com/tech/stl/basic_string.html`

std::stringstream : les flux de chaînes

- std::string ne possède de fonctions pour ajouter des entiers, réels ou booléen directement.

std::stringstream : les flux de chaînes

- std::string ne possède de fonctions pour ajouter des entiers, réels ou booléen directement.
- Les stringstream permettent une édition des chaînes de caractères par les flux :

std::stringstream : les flux de chaînes

- std::string ne possède de fonctions pour ajouter des entiers, réels ou booléen directement.
- Les stringstream permettent une édition des chaînes de caractères par les flux :

Example

```
std::ostringstream st;                                                  1
st << "la␣valeur␣de␣x␣est␣:␣" << 10 << "␣pixels,"                      2
   << "alors␣que␣y␣vaut␣:␣" << 0.345 << "␣points,"                     3
   << "x␣==␣10␣?␣:␣" << std::boolalpha << (x == 10) << ".\n"           4
                                                                        5
std::string chaine(st.str()); // transforme en std::string             6
std::cout << chaine;                                                    7
printf("La␣chaine␣:␣%s", chaine.c_str());                               8
```

Plan

std::vector

Les tableaux de type C existe en C++ :

- Les tableaux alloués dans la pile :

```
double tab1[1000];
```
1

std::vector

Les tableaux de type C existe en C++ :

- Les tableaux alloués dans la pile :

```
double tab1[1000];                                    1
```

- Alloués dans le tas, ne pas oublier la destruction :

```
double* tab2 = new double[1000];                      1
delete[] tab2;                                        2
```

std::vector

Les tableaux de type C existe en C++ :

- Les tableaux alloués dans la pile :

```
double tab1[1000];                                    1
```

- Alloués dans le tas, ne pas oublier la destruction :

```
double* tab2 = new double[1000];                      1
delete[] tab2;                                        2
```

- Alloués dans le tas à deux dimensions :

```
double** tab3 = new double*[1000];                    1
for (size_t i = 0; i < 1000; ++i)                     2
    tab3[i] = new double[1000];                       3
                                                      4
for (size_t i = 0; i < 1000; ++i)                     5
    delete[] tab3[i];                                 6
delete[] tab3;                                        7
```

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.
- La classe std::vector est un très bon remplacement :

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.
- La classe std::vector est un très bon remplacement :
  - Gestion automatique de la mémoire, allocation, réallocation et destruction.

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.
- La classe std::vector est un très bon remplacement :
  - Gestion automatique de la mémoire, allocation, réallocation et destruction.
  - Coût de $O(1)$ pour l'accès aléatoire et la suppression en fin de tableau.

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.
- La classe std::vector est un très bon remplacement :
  - Gestion automatique de la mémoire, allocation, réallocation et destruction.
  - Coût de $O(1)$ pour l'accès aléatoire et la suppression en fin de tableau.
  - Coût de $O(n)$ pour l'ajout ou la suppression d'une case.

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.
- La classe std::vector est un très bon remplacement :
  - Gestion automatique de la mémoire, allocation, réallocation et destruction.
  - Coût de $O(1)$ pour l'accès aléatoire et la suppression en fin de tableau.
  - Coût de $O(n)$ pour l'ajout ou la suppression d'une case.
- Exemple :

std::vector

- Gestion des tableaux C est problématique dans le sens où il faut faire très attention aux accès, à l'allocation, désallocation etc.
- La classe std::vector est un très bon remplacement :
  - Gestion automatique de la mémoire, allocation, réallocation et destruction.
  - Coût de $O(1)$ pour l'accès aléatoire et la suppression en fin de tableau.
  - Coût de $O(n)$ pour l'ajout ou la suppression d'une case.
- Exemple :

Example

```
std::vector < double > tab2(1000);                    1
std::vector < std::vector > tab3(1000);               2
for (size_t i = 0; i < 1000; ++i)                     3
    tab3[i].resize(1000);                             4
```

std::vector

Les fonctions membres :

- operator[](int) et at(int) les opérateurs d'accès aléatoires aux éléments, le deuxième lève une exception si l'entier passé en paramètre sort du tableau.

Example

```
std::vector < double > v(1000, 1.0);                    1
v.push_back(2.0);                                       2
```

std::vector

Les fonctions membres :

- operator[](int) et at(int) les opérateurs d'accès aléatoires aux éléments, le deuxième lève une exception si l'entier passé en paramètre sort du tableau.
- size_t size() et resize(size_t) et reserve(size_t) les fonctions pour, respectivement, connaître la taille d'un tableau, modifier sa taille ou préallouer une zone mémoire.

Example

```
std::vector < double > v(1000, 1.0);                    1
v.push_back(2.0);                                       2
```

std::vector

Les fonctions membres :

- operator[](int) et at(int) les opérateurs d'accès aléatoires aux éléments, le deuxième lève une exception si l'entier passé en paramètre sort du tableau.
- size_t size() et resize(size_t) et reserve(size_t) les fonctions pour, respectivement, connaître la taille d'un tableau, modifier sa taille ou préallouer une zone mémoire.
- push_back(T), pop_back(), les fonction d'ajout et de suppression d'élément en fin de tableau (coût $O(1)$).

Example

```
std::vector < double > v(1000, 1.0);                    1
v.push_back(2.0);                                       2
```

std::vector, les itérateurs

Les itérateurs permettent de manipuler plus facilement les éléments des conteneurs de la STL :

Example

```
std::vector < std::string > t(200);                                    1
for (std::vector < int >::iterator it = t.begin();                     2
  it != t.end(); ++it)                                                 3
    std::cout << (*it) << `` '';                                       4
                                                                       5
std::vector < int >::iterator fd;                                      6
                                                                       7
fd = std::find(t.begin(), t.end(), ``debian'');                        8
                                                                       9
if (fd != t.end())                                                     10
    v.erase(fd);                                                       11
```

Plan

std::list

- Une liste doublement chaînée avec un coût en $O(1)$ pour l'ajout, la suppression n'importe où dans la liste.

std::list

- Une liste doublement chaînée avec un coût en $O(1)$ pour l'ajout, la suppression n'importe où dans la liste.
- push_back(T), push_front(T), pop_back() et pop_front() les fonctions d'ajout en début et fin de liste et les fonctions de suppression en début et fin de liste.

std::list

- Une liste doublement chaînée avec un coût en $O(1)$ pour l'ajout, la suppression n'importe où dans la liste.
- push_back(T), push_front(T), pop_back() et pop_front() les fonctions d'ajout en début et fin de liste et les fonctions de suppression en début et fin de liste.

std::list

- Une liste doublement chaînée avec un coût en $O(1)$ pour l'ajout, la suppression n'importe où dans la liste.
- push_back(T), push_front(T), pop_back() et pop_front() les fonctions d'ajout en début et fin de liste et les fonctions de suppression en début et fin de liste.

Example

```
std::list < int > l;                                    1
l.push_back(5);                                         2
l.push_front(3);                                        3
                                                        4
std::cout << l.front() == 3;                            5
                                                        6
std::list < int >::iterator it;                         7
it = std::find(l.begin(); l.end(); 3);                  8
                                                        9
std::cout << it != l.end();                             10
```

Plan

Les map : tableau associatif

Cette classe est une table associant une clé à une valeur. Elle est définie en fournissant deux types. À partir de la clé nous pouvons récupérer la valeur avec un coût d'ordre $O(log(N))$.

Example

```
map < double , string > m;
m[0.1] = "hello ";
m[0.5] = "world ";
m[0.6] = "!";

map < double, string >::iterator it = m.begin();
for (; it != m.end(); ++it)
    cout << (*it).first << " " << (*it).second << "\n";

if (m.find(1.0) == m.end()) cout << "1.0 n'existe pas";
cout << m[1.0] << "\n";
if (m.find(1.0) == m.end()) cout << "1.0 existe";
```

Les liens

Une petite liste de liens :

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

Les liens

Une petite liste de liens :

- stl http://www.sgi.com/tech/stl/
- C++ http://www.cplusplus.com/

Les liens

Une petite liste de liens :

- stl http://www.sgi.com/tech/stl/
- C++ http://www.cplusplus.com/
- faq C++ http://www.parashift.com/c++-faq-lite/index.html

Les liens

Une petite liste de liens :

- stl http://www.sgi.com/tech/stl/
- C++ http://www.cplusplus.com/
- faq C++ http://www.parashift.com/c++-faq-lite/index.html
- gtk_{mm} http://www.gtkmm.org

Les liens

Une petite liste de liens :

- stl http://www.sgi.com/tech/stl/
- C++ http://www.cplusplus.com/
- faq C++ http://www.parashift.com/c++-faq-lite/index.html
- gtk_{mm} http://www.gtkmm.org
- gtk_{mm} le livre : http://www.gtkmm.org/gtkmm2/docs/tutorial/html/index.html

Les liens

Une petite liste de liens :

- stl http://www.sgi.com/tech/stl/
- C++ http://www.cplusplus.com/
- faq C++ http://www.parashift.com/c++-faq-lite/index.html
- gtk_{mm} http://www.gtkmm.org
- gtk_{mm} le livre : http://www.gtkmm.org/gtkmm2/docs/tutorial/html/index.html
- *gtk+* http://www.gtk.org