Programmation C++

Template - STL

ING2-GSI

CY Tech

2023-2024

Template

Template

- Les templates permettent à une fonction ou une classe d'utiliser les types différents :
  - Performance : code compact et efficace
  - Temps de codage
  - Clarté
- Programmation généréque :
  - Algorithmes génériques
  - Polymorphisme paramétrique

Ex1 : Fonction template

Exemple.cpp

```cpp
#include <iostream>

template <typename T>          // ou template <class T>
T mini(const T & a, const T & b) {
    return (a < b ? a : b);
}

int main() {
    double dmin, d1{5.1}, d2{9.5};
    int imin, i1{9},i2{4};
    Fraction f1(1,2), f2(1,3);
    dmin = mini(d1, d2) ;
    imin = mini<int>(i1,i2) ;
    std::cout << mini(f1,f2) ; // operator<
    return 0;
}
```

Ex2 : Fonction template

AutreExemple.cpp

```
template<typename T1, typename T2>
T2 calculerMoyenne(T1 tableau[], int taille) {
    T2 somme = 0;
    for(int i = 0; i<taille; ++i)
        somme += tableau[i];
    return somme/taille;
}

int main() {
    ...
    calculerMoyenne<int,double>(tab,5);
    ...
}
```

Votre première classe Vector

Vector.hpp

```
class Vector {
private:
    double* elem;
    unsigned int sz;
public:
    Vector(unsigned int s);
    ~Vector();
    double & operator[](unsigned int i);
    unsigned int size() const;
};
```

Ex3 : Template : Vector

Vector.hpp

```
template<typename T>
class Vector {
private:
    T* elem;
    unsigned int sz;
public:
    Vector(unsigned int s);
    ~Vector() { delete[] elem; }
    // ... copy and move operations ...
    T& operator[](unsigned int i);
    const T& operator[](unsigned int i) const;
    unsigned int size() const { return sz; }
};
```

Ex3 : Template : Vector

Vector.hpp

```
template<typename T>
Vector<T>::Vector(unsigned int s) {
    elem = new T[s];
    sz = s;
}

template<typename T>
const T& Vector<T>::operator[](unsigned int i) const {
    if (size()<=i)
        throw out_of_range{"Vector::operator[]"};
    return elem[i];
}
```

Attention !

Implémentation dans le fichier d'en-tête : **Vector.hpp**

Ex3 : Template : Vector

main.cpp

```
int main() {
    Vector<char> vc(314);
    Vector<string> vs(42);
    Vector<list<int>> vli(74);
    //...
}
```

STL

Standard Template Librairy (STL)

"Don't reinvent the wheel !" - Stroustrup

- Algorithmes et structures de données fondamentaux
- Librairies bien codées, testées et optimisées

Standard Template Librairy (STL)

- Conteneurs : vector, list, set, map, stack, ...
- Itérateurs
- Algorithmes : insertion, suppression, recherche, tri, numériques (accumulate, inner_product, etc) ...
- Foncteur

STL - Conteneur

- Les conteneurs sont des classes permettant de représenter les structures de donnés les plus répandues
- 2 catégories de conteneurs :
  - Conteneurs séquentiels : les éléments sont ordonnés. On peut parcourir le conteneur suivant cet ordre et insérer ou supprimer un élément à un endroit explicitement choisi : vector, list, stack, ...
  - Conteneurs associatifs : les éléments sont identifiés par une clé et ordonnés selon celle-ci : set, multiset, map, multimap
- Fonctionnalités communes :
  - `int size() const;`
  - `bool empty() const;`
  - ...

Ex4 : STL - Conteneur séquentiel **list**

- Déclaration :

```
list <Type> l;
```

- Quelques méthodes : ▸ Liste compl`ete

```
void push_front(const Type &);
void pop_front();
void remove(const Type &);
```

```
#include <list>

list<int> l;
for (int i=0; i<5; i++)
    l.push_back((10+2*i)% 5 +i);      // {0,3,1,4,2}
l.sort();                             // {0,1,2,3,4}
l.reverse();                          // {4,3,2,1,0}
cout << l.front() << " " << l.back() << endl;
```

Ex4 : STL - Conteneur associatif **map**

- Déclaration :

  map<TypeClef, TypeValeur> m;

- Méthodes : ▸ Liste compl`ete

```
#include <map>

map <string, string> repertoire;
repertoire["Jean Martin"] = "01.02.03.04.05";
repertoire["Francois Martin"] = "02.03.04.05.06"
;repertoire["Louis Dupont "] = "03.04.05.06.07";
repertoire.insert(pair<string,string>("Louis Martin",
"04.05.06.07.08"));
```

Ex4 : STL - Conteneur associatif

- unordered_map, unordered_set :
  - table de hachage
  - recherche, insertion, suppression : O(1)
- map, set :
  - arbre binaire de recherche équilibré (red-black, AVL, ...)
  - recherche, insertion, suppression : O(log(*n*))

Articles sur la comparaison : Map vs. UnsortedMap | Set vs. UnsortedSet

Ex4 : STL - Iterator

Définition

Un itérateur est un objet qui peut pointer sur un élément dans une collection, et qui a la capacité de parcourir les éléments de la collection en utilisant un ensemble d'opérateurs (incrémentation ++, déréférence *, ...).

```
#include <vector>

int myints[] = {32,71,12,45,26,80,53,33};
// the iterator constructor can also be used to
// construct from arrays
vector<int> myvector (myints, myints+8);
for (vector<int>::iterator it=myvector.begin();
    it!=myvector.end(); ++it) {
        cout << ' ' << *it;
}
```

Ex4 : STL - auto

```
#include <vector>

int myints[] = {32,71,12,45,26,80,53,33};

vector<int> myvector (myints, myints+8);

for (auto it=myvector.begin(); it!=myvector.end();
    ++it) {
        cout << ' ' << *it;
}
```

Le compilateur va trouver le bon type

Ex4 : STL - Algorithmes

Définition

Un algorithme est une fonction template qui peut être utilisée dans une collection d'éléments, en utilisant des itérateurs.

▸ Liste des algorithmes

```
#include <algorithm>
#include <vector>

vector <int> v;
...
//Compter le nombre d'occurrences d'une valeur
int n = count(v.begin(), v.end(),1);
//Melanger aleatoirement les valeurs
random_shuffle(myvector.begin(), myvector.end());
```

Ex4 : STL - Foncteur (fonction objet)

Définition

Les foncteurs sont des objets possédant une surcharge de l'**operator()**. Ils sont souvent utilisés comme arguments de fonction, comme des prédicats ou des fonctions de comparaison dans les algorithmes.

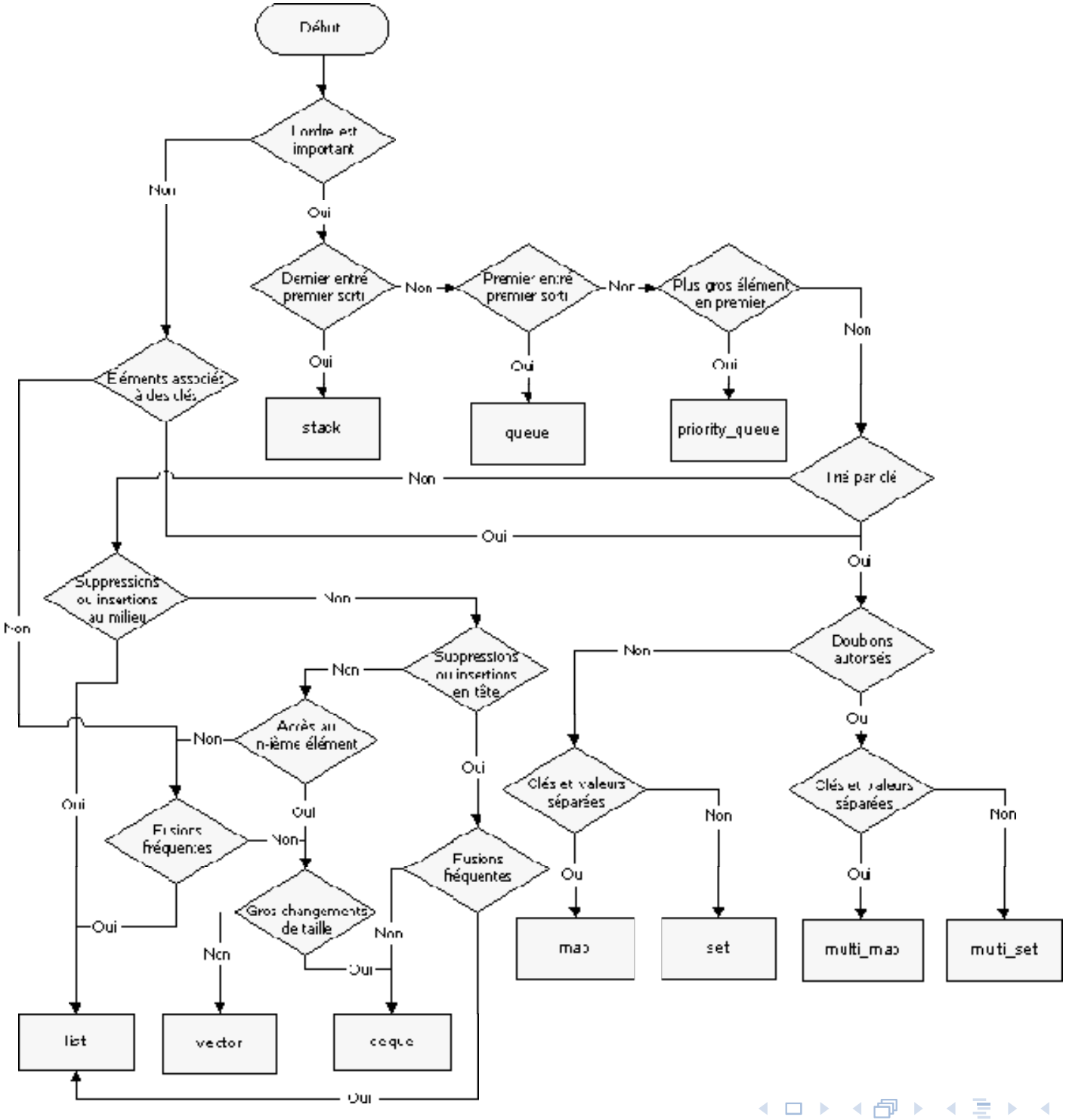
```
template <typename T>
class plusGrand {
public :
    bool operator() (const T& x, const T& y) const {
        return x > y;
    }
};

int numbers[]={20,40,50,10,30};
sort(numbers, numbers+5, plusGrand<int >());
```

Ex4 - STL : lambda-expression & for-each

```
int vector<int> v(6)={10, 20,40,50,10,30}
;

for_each(v.begin(), v.end(),
    [](int i) {cout << i << endl;});
```


Début
L'ordre est important
Non
Oui
Dernier entré premier sorti
Non
Premier entré premier sorti
Non
Plus gros élément en premier
Non
Oui
Oui
Oui
stack
queue
priority_queue
Eléments associés à des clés
Non
Trié par clé
Oui
Oui
Suppressions ou insertions au milieu
Non
Non
Suppressions ou insertions en tête
Non
Doublons autorsés
Non
Ou
Accès au n-ième élément
Non
Oui
Oui
Clés et valeurs séparées
Non
Clés et valeurs séparées
Non
Fusions fréquentes
Non
Oui
Fusions fréquentes
Oui
Ou
Ou
Oui
Oui
Non
Gros changements de taille
Non
Oui
Non
map
set
multi_map
muti_set
list
vector
deque
Oui