code a proper sort yourself

std::sort()

R2.01
DEV2

Développement orienté objets

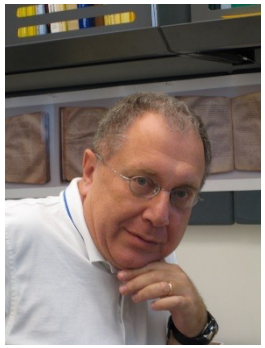
Etienne Carnovali **Pétra Gomez**

Farid Ammar-Boudjelal Jean-Michel Bohé Rouaa Wannous

Standard Template Library

- Un peu d'histoire
- C++ Standard Library
- Les modèles ou patrons - *templates*
- Les conteneurs - *containers*
- Les itérateurs - *iterators*
- Les algorithmes - *algorithm*

Un peu d'histoire



Alexander STEPANOV

Meng LEE

Standard Library C++98 03 11 14 17 20 23

C++ Standard Library

- C Standard library
- string
- flux : E/S, fichiers, …
- algorithmes
- STL
- exceptions

→ using namespace std ;

https://en.cppreference.com/w/cpp/header

Les modèles …

- On souhaite écrire une seule fonction de traitement quel que soit le type de donnée
  - surcharge

```
void swapValues(int& a, int& b){
  int c=a;
  a=b;
  b=c;
}

void swapValues(double& a, double& b){
  double c=a;
  a=b;
  b=c;
}
```

```
int a=12, b=9 ;
cout << "a= " << a << " b= " << b << endl;
swapValues(a, b);
cout << "a= " << a << " b= " << b << endl;

double a=12, b=9 ;
cout << "a= " << a << " b= " << b << endl;
swapValues(a, b);
cout << "a= " << a << " b= " << b << endl;
```

… au top !

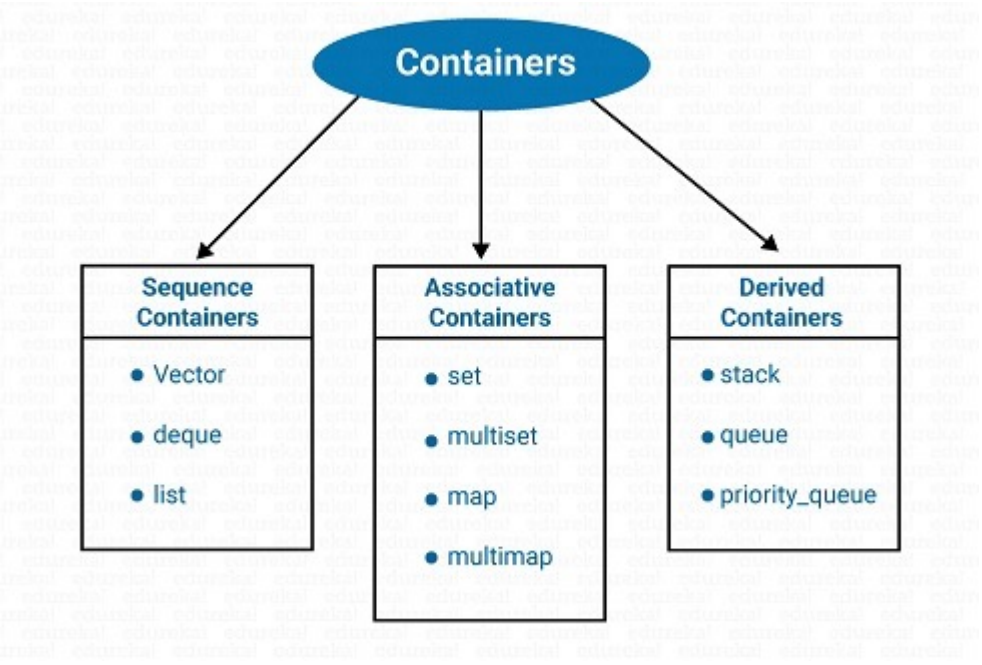
- On souhaite écrire **une seule fonction** de traitement quel que soit le type de donnée
  - généricité

```
template <typename T>
void swapValues(T& a, T& b){
  T c=a;
  a=b;
  b=c;
}
```

```
string strA="patron", strB="modèle";
swapValues(strA, strB);
cout << "strA= " << strA << " strB= " << strB << endl;
```

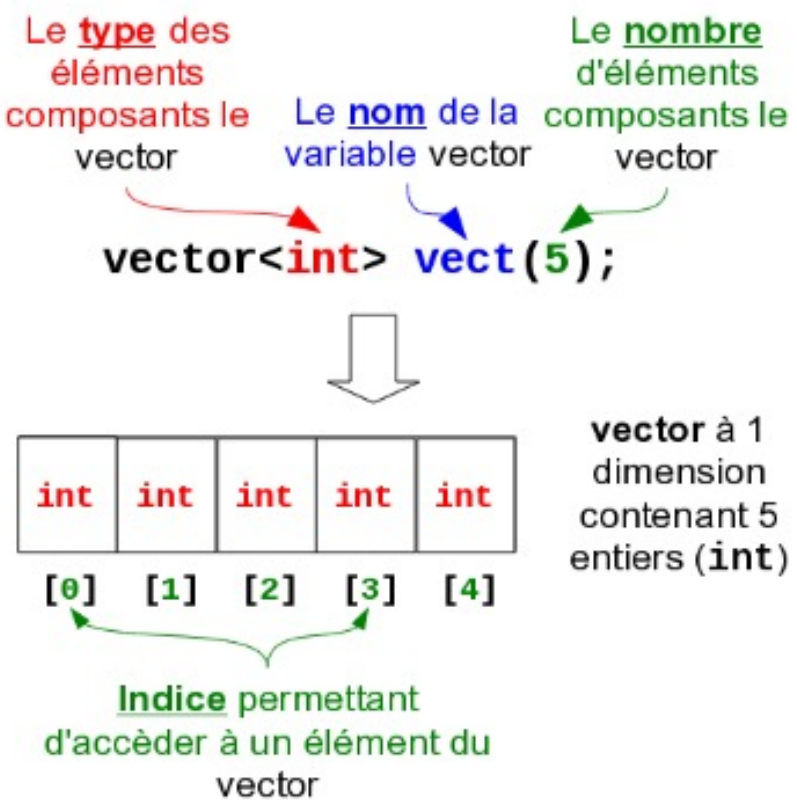
Les conteneurs

- Conteneurs de séquence
  - tableaux (statiques et dynamiques),
  - listes,
  - dèques.
- Conteneurs associatifs
  - ensembles,
  - multi-ensembles,
  - tables,
  - multi-tables
- Conteneurs adaptateurs
  - pile,
  - file,
  - file prioritaire

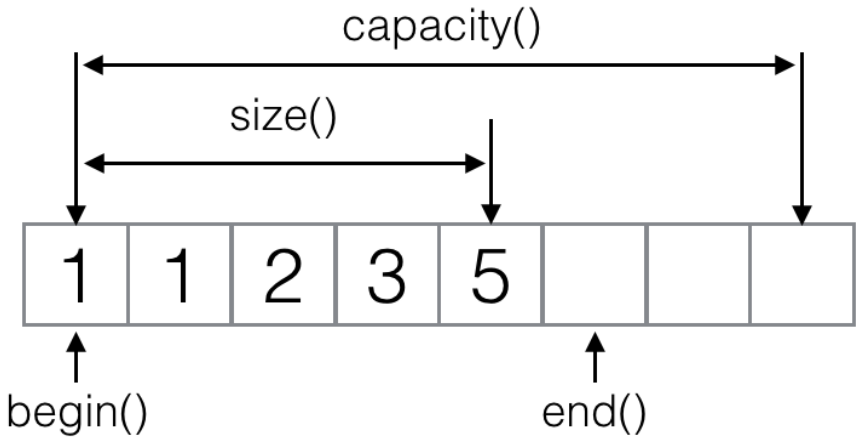


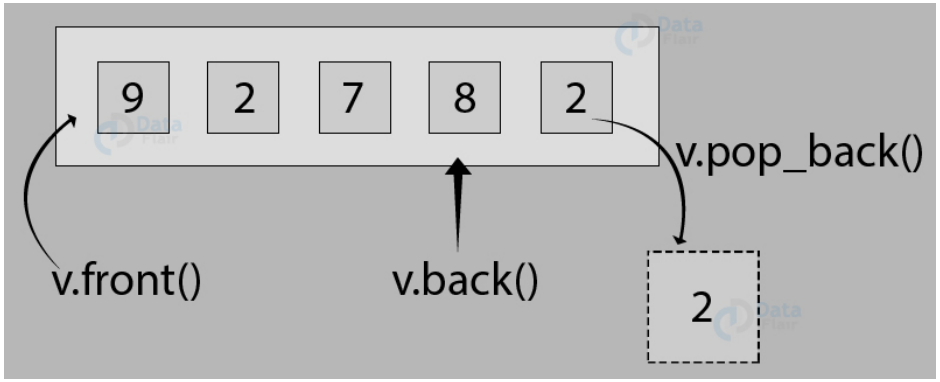
Les tableaux dynamiques <vector>

- conteneur séquentiel
- éléments contigus
- accès direct
- ajout ou suppression à la fin



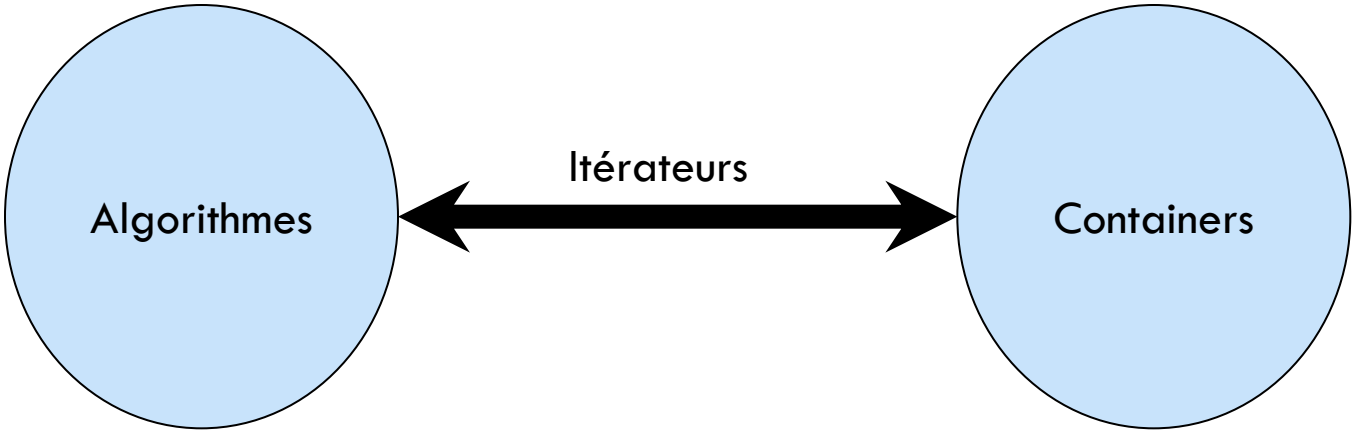
Les tableaux dynamiques <vector>





```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
  vector<int> vect(5); // un vecteur de 5 entiers

  vect[0] = 1; // accès direct à l'indice 0 pour affecter la valeur 1
  vect[1] = 2; // accès direct à l'indice 1 pour affecter la valeur 2

  // augmente et diminue la taille du vector
  vect.push_back( 6 ); // ajoute l'entier 6 à la fin
  vect.push_back( 7 ); // ajoute l'entier 7 à la fin
  vect.push_back( 8 ); // ajoute l'entier 8 à la fin
  vect.pop_back(); // enleve le dernier élément et supprime l'entier 8

  cout << "Le vecteur vect contient " << vect.size() << " entiers : \n";

  // utilisation d'un indice pour parcourir le vecteur vect
  for(int i=0;i<vect.size();i++)
      cout << "vect[" << i << "] = " << vect[i] << '\n';
  cout << '\n';

  return 0;
}
```

The schema par STEPANOV et LEE

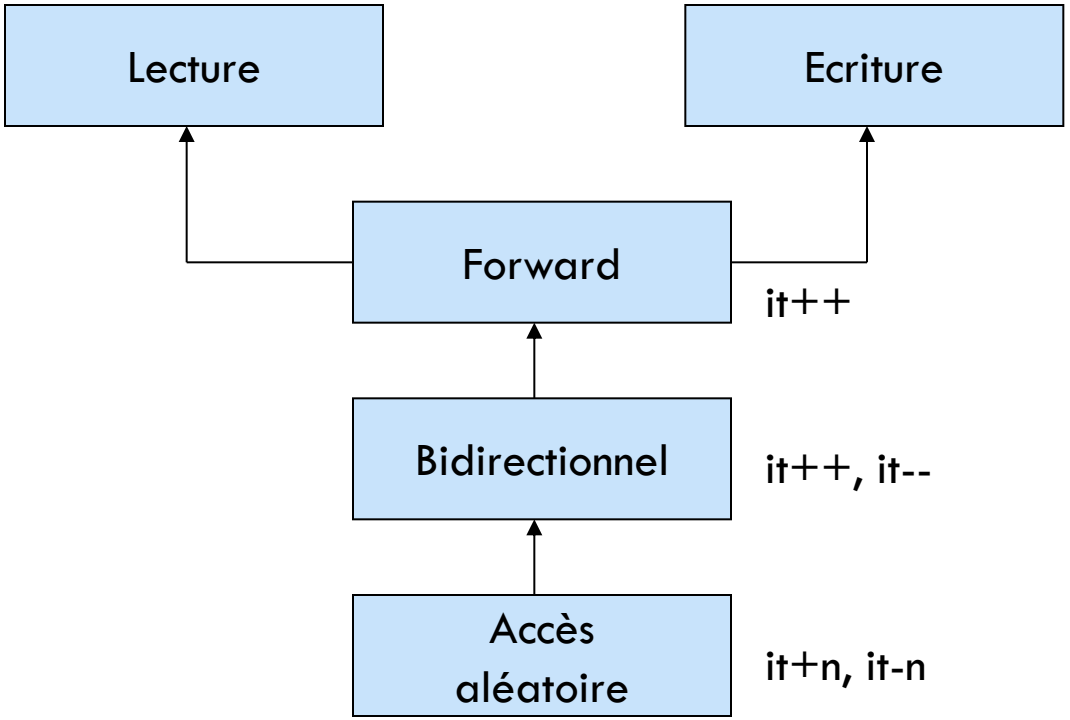


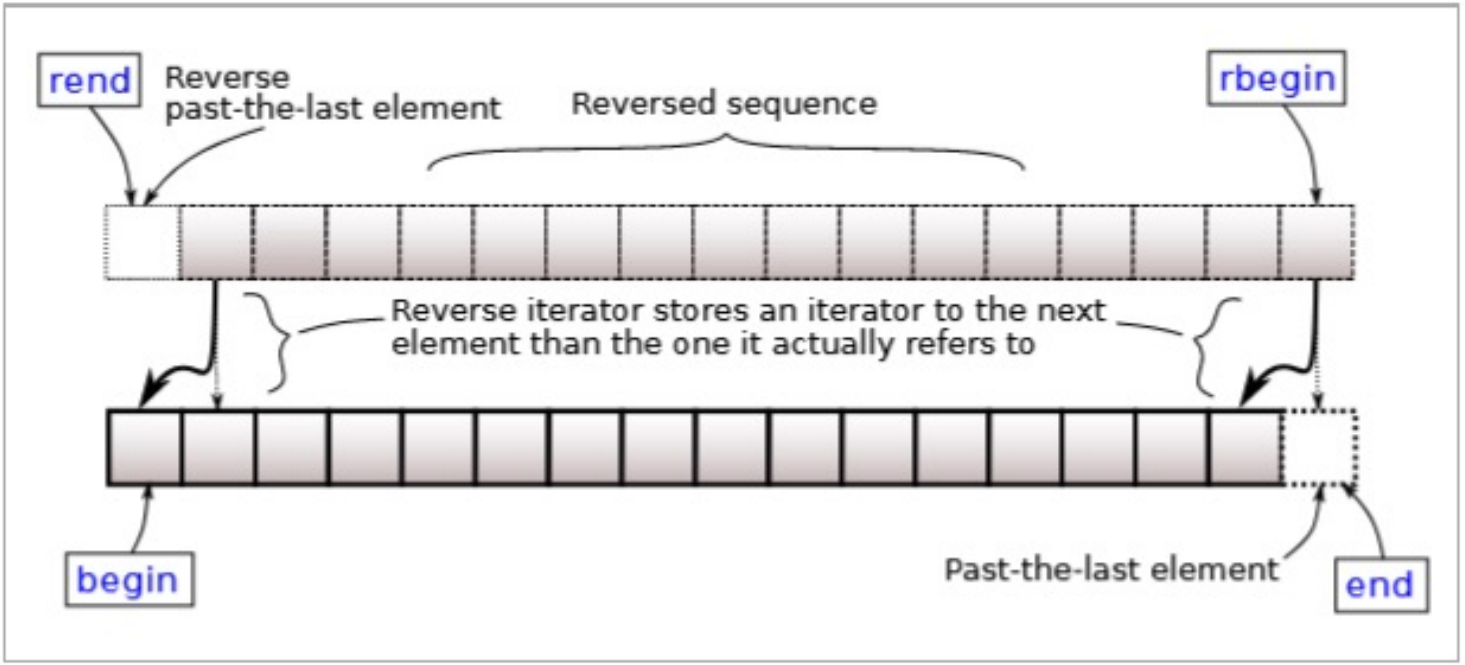
- Les algorithmes sont indépendants du type de données stockées
- Connecter des algorithmes à des structures de données, par le biais des itérateurs.

Les itérateurs

<iterator>

- généralisation des pointeurs
- déplacement par itération
- parcourt les éléments d'un conteneur.
- Hiérarchie des itérateurs :



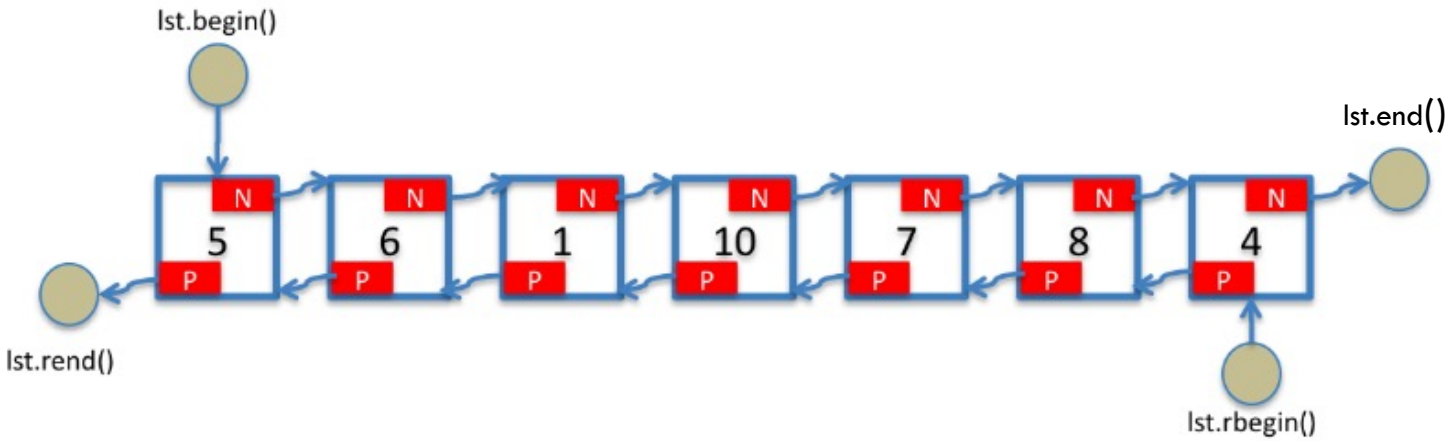



```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  vector<int> v2(4, 100); // un vecteur de 4 entiers initialisés avec la valeur 100
  cout << "Le vecteur v2 contient " << v2.size() << " entiers : ";
  // utilisation d'un itérateur pour parcourir le vecteur v2
  for (vector<int>::iterator it = v2.begin(); it != v2.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';
  return 0;
}
```

Les listes

<list>

- liste doublement chaînée
- éléments non contigus



```cpp
#include <iostream>
#include <list>

using namespace std;

int main()
{
  list<int> lst; // une liste vide

  lst.push_back( 5 );
  lst.push_back( 6 );
  lst.push_back( 1 );
  lst.push_back( 10 );
  lst.push_back( 7 );
  lst.push_back( 8 );
  lst.push_back( 4 );
  lst.push_back( 5 );
  lst.pop_back(); // enleve le dernier élément et supprime l'entier 5

  cout << "La liste lst contient " << lst.size() << " entiers : \n";
```

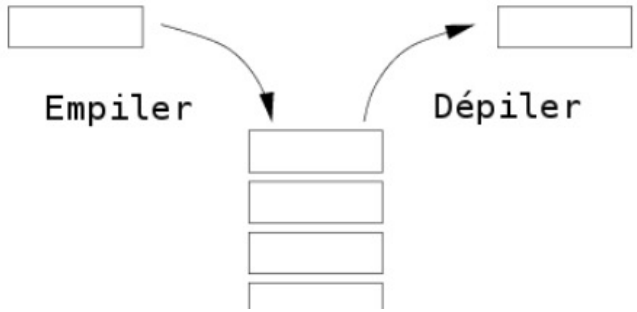
```
  // utilisation d'un itérateur pour parcourir la liste lst
  for (list<int>::iterator it = lst.begin(); it != lst.end(); ++it)
    cout << ' ' << *it;
  cout << '\n';

  // afficher le premier élément
  cout << "Premier element : " << lst.front() << '\n';
  // afficher le dernier élément
  cout << "Dernier element : " << lst.back() << '\n';

  // parcours avec un itérateur en inverse
  for ( list<int>::reverse_iterator rit = lst.rbegin(); rit != lst.rend(); ++rit )
  {
      cout << ' ' << *rit;
  }
  cout << '\n';

  return 0;
}
```

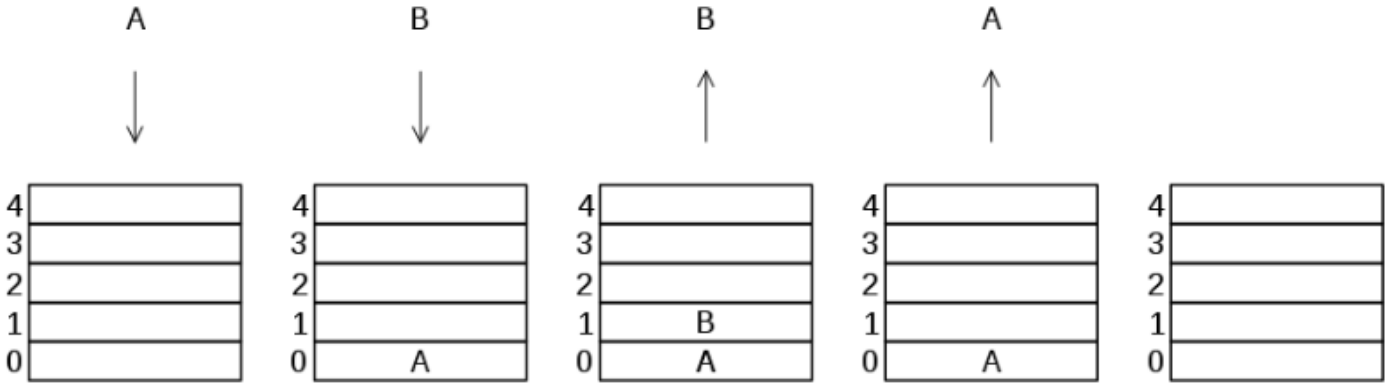
```
La liste lst contient 7 entiers :
 5 6 1 10 7 8 4
Premier element : 5
Dernier element : 4
 4 8 7 10 1 6 5
```

Les piles

<stack>

- LIFO



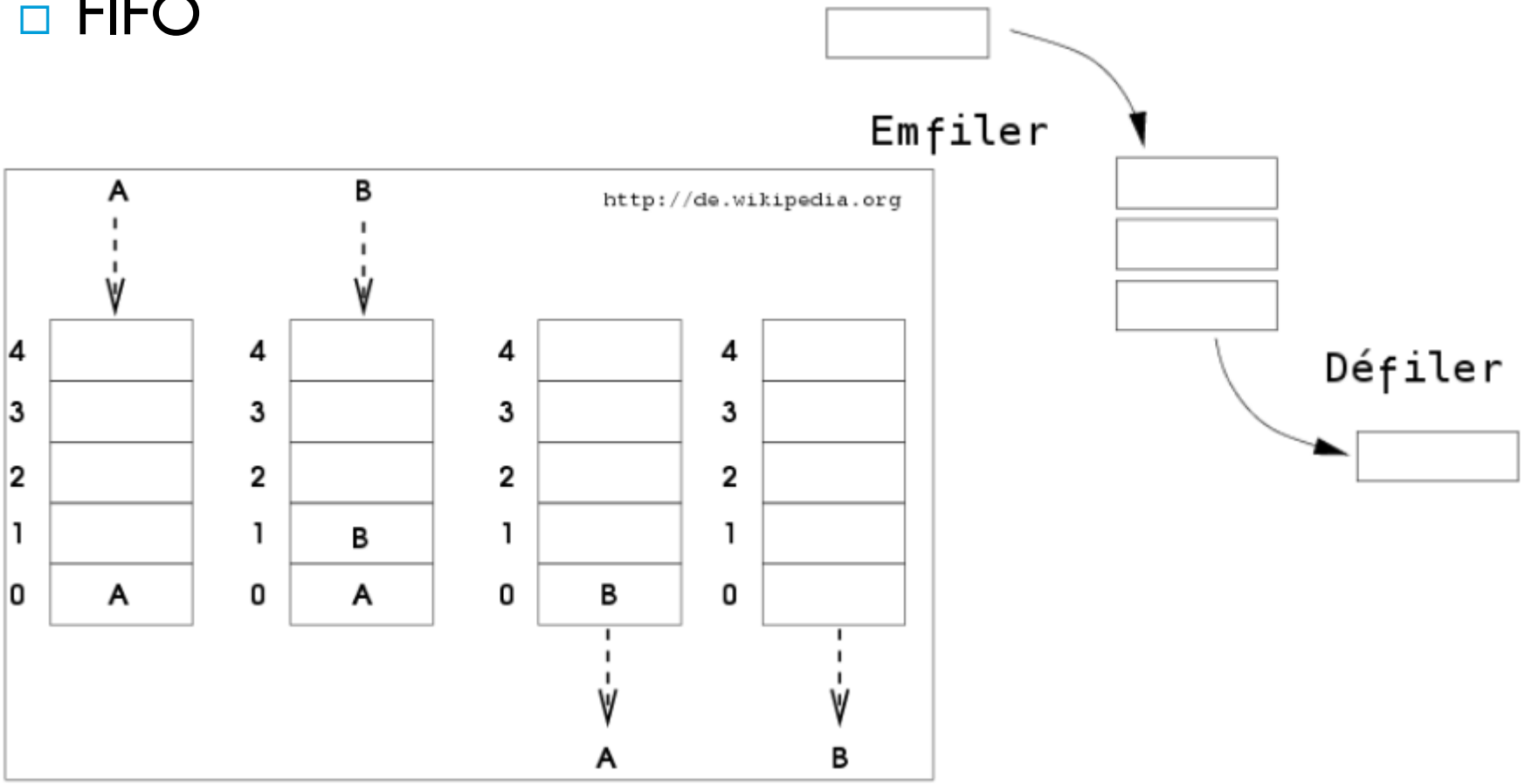


```cpp
#include <iostream>
#include <stack>
using namespace std;

/* Les opérations de base :
    void pile.push(valeur); // Empiler
    T    pile.top();        // Retourne la valeur du haut de la pile
    void pile.pop();        // Dépiler
    bool pile.empty();      // Retourne true si la pile est vide sinon false
    void pile.clear();      // Vider la pile

   Lien : http://www.cplusplus.com/reference/stack/stack/ */

int main()
{
    stack<int> pile;
    pile.push(4);
    pile.push(2);
    pile.push(1);

    cout << "Taille de la pile : " << pile.size() << endl;
    while (!pile.empty())
    {
        cout << pile.top() << endl;
        pile.pop();
    }
    cout << "Taille de la pile : " << pile.size() << endl;

    return 0;
}
```

Les files

<queue>

- FIFO



```cpp
#include <iostream>
#include <queue>

using namespace std;

/* Les opérations de base :
    void file.push(valeur); // Empiler
    T    file.front();     // Retourne la valeur la plus "ancienne"
    T    file.back();      // Retourne la valeur la moins "ancienne"
    void file.pop();       // Dépiler la valeur la plus "ancienne"
    bool file.empty();     // Retourne true si le tas est vide sinon false
    void file.size();      // Retourne la taille du tas

   Lien : http://www.cplusplus.com/reference/queue/queue/ */

int main()
{
    queue<int> file;

    file.push(1);
    file.push(4);
    file.push(2);

    cout << "Taille de la file : " << file.size() << endl;
    while (!file.empty())
    {
        cout << file.front() << endl;
        file.pop();
    }
    cout << "Taille de la file : " << file.size() << endl;
    return 0;
}
```

Les tables associatives

<map>

- association clé ←→ donnée
- un type pour la clé
- un type pour la donnée
- une paire

```cpp
#include <iostream>
#include <iomanip>
#include <map>
#include <string>

using namespace std;

int main()
{
  map<string,unsigned int> nbJoursMois;

  nbJoursMois["janvier"] = 31;
  nbJoursMois["février"] = 28;
  nbJoursMois["mars"] = 31;
  nbJoursMois["avril"] = 30;
  //...

  cout << "La map contient " << nbJoursMois.size() << " elements : \n";
  for (map<string,unsigned int>::iterator it=nbJoursMois.begin(); it!=nbJoursMois.end(); ++
      it)
  {
    cout << it->first << " -> \t" << it->second << endl;
  }

  cout << "Nombre de jours du mois de janvier : " << nbJoursMois.find("janvier")->second <<
      '\n';
```

```
La map contient 4 elements :
avril ->        30
février ->      28
janvier ->      31
mars ->         31
Nombre de jours du mois de janvier : 31
```

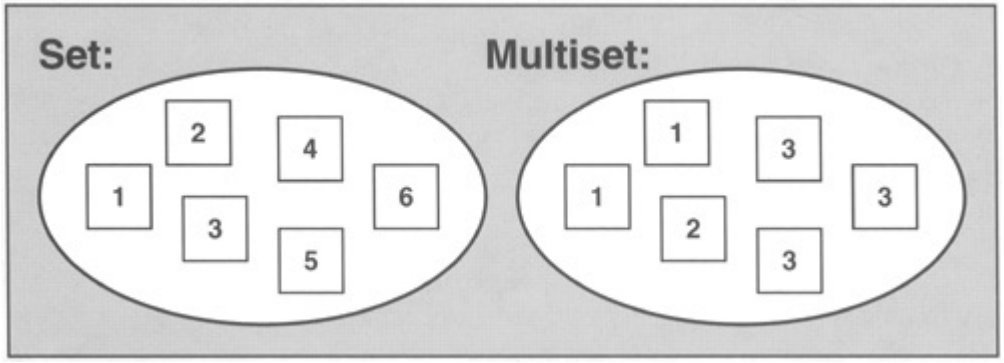
```
  // affichage du mois de janvier
  cout << "Janvier : \n" ;
  for (int i=1; i <= nbJoursMois["janvier"]; i++)
  {
      cout << setw(3) << i;
      if(i%7 == 0)
         cout << endl;
  }
  cout << endl;

  return 0;
}
```

```
Janvier :
  1  2  3  4  5  6  7
  8  9 10 11 12 13 14
 15 16 17 18 19 20 21
 22 23 24 25 26 27 28
 29 30 31
```

Les ensembles

<set>

- ensemble ordonné sans doublons
- multiset avec doublons possibles
- value=key



```cpp
#include <iostream>
#include <set>
using namespace std;

int main()
{
  int desEntiers1[] = {75, 23, 65, 42, 13, 100}; // non ordonné
  int desEntiers2[] = {75, 23, 75, 23, 13}; // non ordonné avec des doublons
  set<int> ensemble1 (desEntiers1, desEntiers1+6); // the range (first,last)
  set<int> ensemble2 (desEntiers2, desEntiers2+5); // the range (first,last)

  cout << "L'ensemble set 1 contient :";
  for (set<int>::iterator it=ensemble1.begin(); it!=ensemble1.end(); ++it)
  {
    cout << ' ' << *it;
  }

  cout << endl;

  cout << "L'ensemble set 2 contient :";
  for (set<int>::iterator it=ensemble2.begin(); it!=ensemble2.end(); ++it)
  {
    cout << ' ' << *it;
  }

  cout << endl;

  return 0;
}
```

```
L'ensemble set 1 contient : 13 23 42 65 75 100
L'ensemble set 2 contient : 13 23 75
```

Les paires

<pair>

- structure contenant deux éléments éventuellement de types différents
  - first
  - second
- Certains algorithmes de la STL (find par exemple) retournent des paires
  - position de l'élément trouvé
  - un booléen indiquant s'il a été trouvé

```cpp
#include <iostream>
#include <utility>
#include <vector>
#include <list>
#include <map>
#include <set>

using namespace std;

int main()
{
  pair<char,int> c1 = make_pair('B', 2); // coordonnées type "bataille navale"
  pair<char,int> c2 = make_pair('J', 1);

  cout << "Un coup en " << c1.first << '.' << c1.second << endl;

  pair<int,int> p; // position de type "morpion"

  p.first = 1;
  p.second = 2;

  cout << "Un coup en " << p.first << ',' << p.second << endl;
```

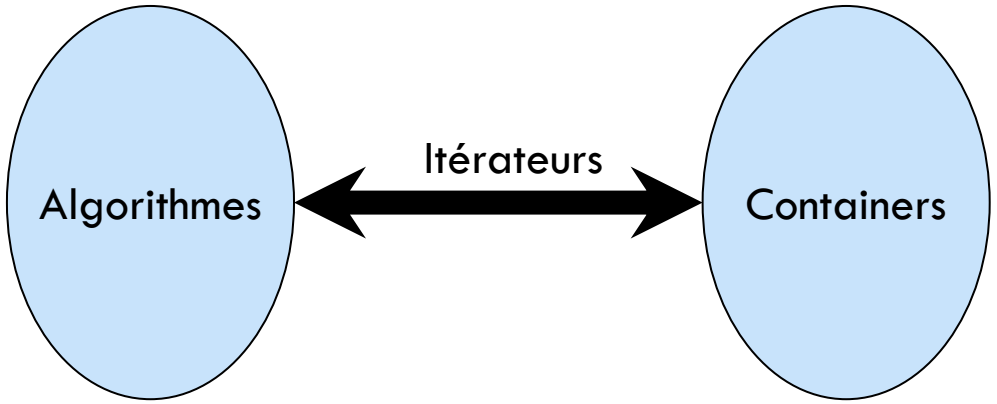
```
vector<pair<char,int> > tableauCoups(2); // ou par exemple : list<pair<char,int> >
    listeCoups;

tableauCoups[0] = c1;
tableauCoups[1] = c2;
cout << "Le vector contient " << tableauCoups.size() << " coups : \n";
for (vector<pair<char,int> >::iterator it=tableauCoups.begin(); it!=tableauCoups.end(); ++
    it)
{
  cout << (*it).first << "." << (*it).second << endl;
}
cout << endl;
```

```
map<pair<char,int>,bool> mapCoups;

mapCoups[c1] = true; // ce coup a fait mouche
mapCoups[c2] = false; // ce coup a fait plouf

cout << "La map contient " << mapCoups.size() << " coups : \n";
for (map<pair<char,int>,bool>::iterator it=mapCoups.begin(); it!=mapCoups.end(); ++it)
{
  cout << it->first.first << "." << it->first.second << " -> \t" << it->second << endl;
}
cout << endl;
```

```
set<pair<char,int> > ensembleCoups;

ensembleCoups.insert(c1);
ensembleCoups.insert(c2);

cout << "L'ensemble set contient " << ensembleCoups.size() << " coups : \n";
for (set<pair<char,int> >::iterator it=ensembleCoups.begin(); it!=ensembleCoups.end(); ++
    it)
{
  cout << (*it).first << "." << (*it).second << endl;
}
cout << endl;
```

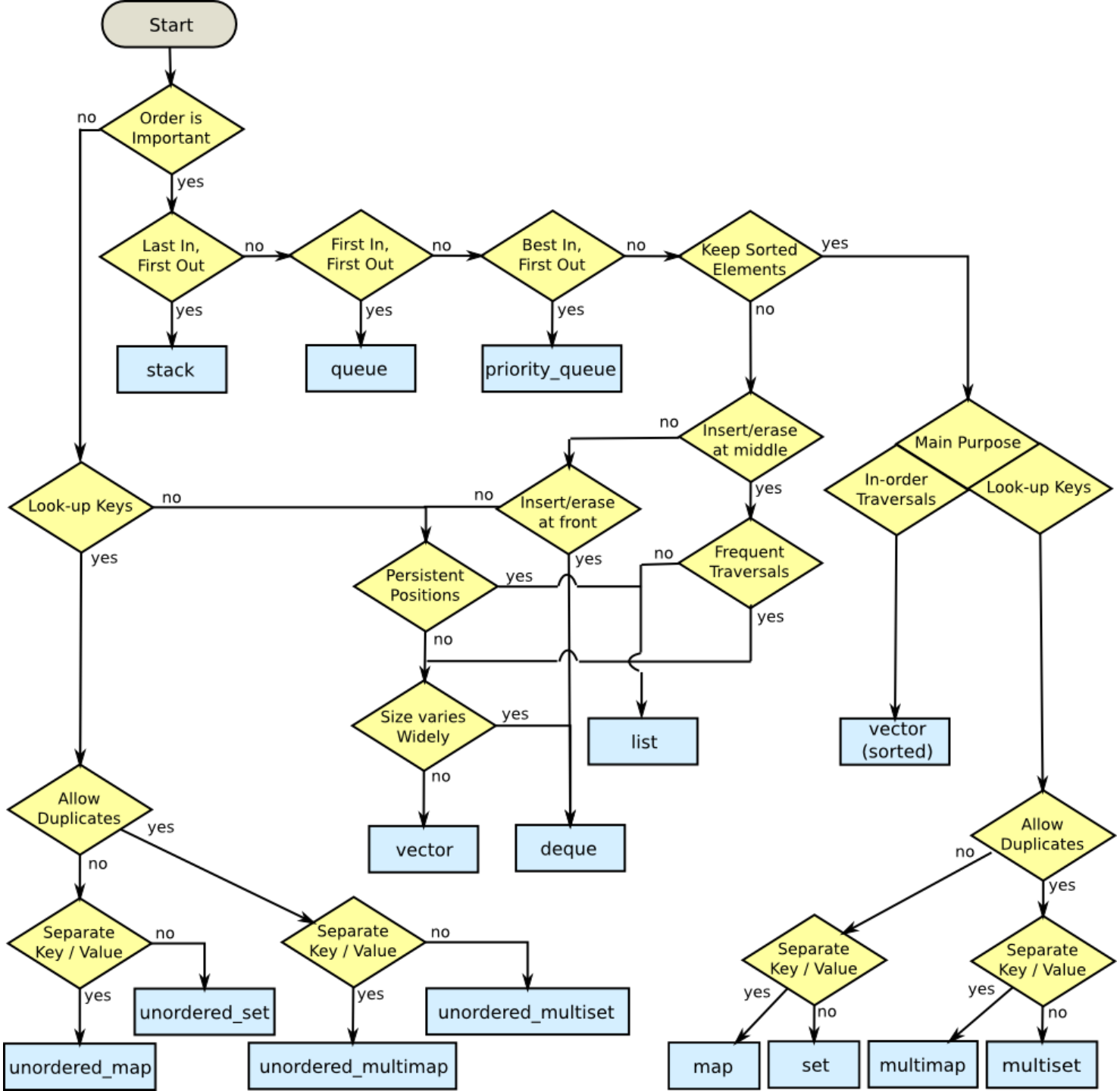
Les algorithmes <algorithm>

- find
- find_if
- for_each
- generate
- random_shuffle
- copy
- sort
- unique

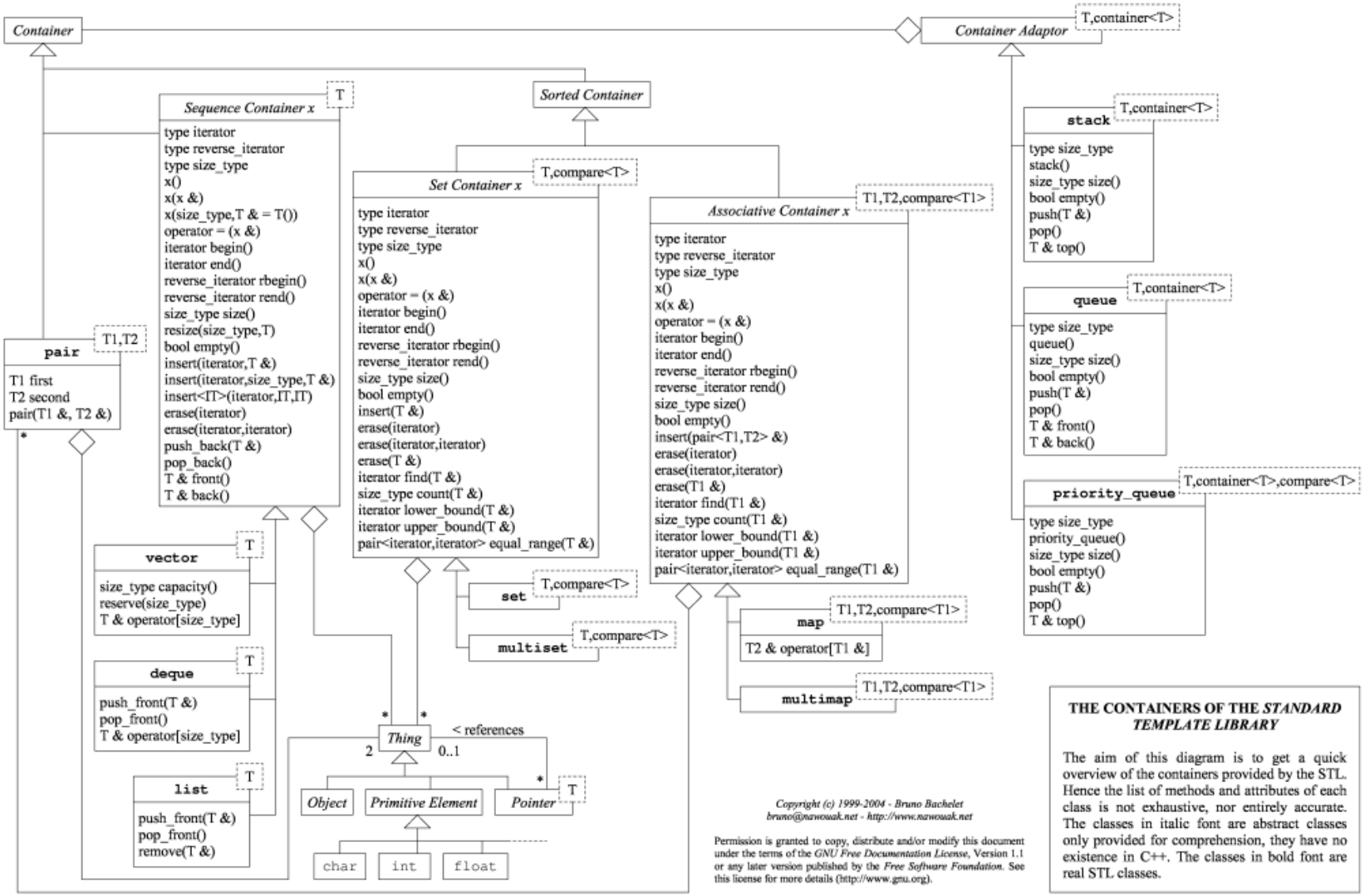


Foncteur


```cpp
#include <vector>
#include <algorithm>
#include <iterator>
#include <iostream>
using namespace std;

// ----------------------------------------------
// function used to generate random numbers
// between 0 and 99
// ----------------------------------------------
int my_generator() {
    return rand() % 100;
}

// ----------------------------------------------
// main function
// ----------------------------------------------
int main() {
    srand(19702013);

    // must define the size of the vector
    vector<int> v(20);

    // call user-defined function
    generate(v.begin(), v.end(), my_generator);

    // print vector contents
    copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
    cout << endl;

    vector<int>::iterator it;

    // find minimum element
    it = min_element(v.begin(), v.end());
    cout << "minimum = " << (*it) << endl;

    // find maximum element
    it = max_element(v.begin(), v.end());
    cout << "maximum = " << (*it) << endl;

    // C++11 minmax_element
    pair<vector<int>::iterator, vector<int>::iterator> result;
    result = minmax_element(v.begin(), v.end());
    cout << "with minmax_elmeent C++11: " << endl;
    cout << "minimum = " << *(result.first) << endl;
    cout << "maximum = " << *(result.second) << endl;


    return 0;
}
```


Start
Order is Important
no
yes
Last In, First Out
First In, First Out
Best In, First Out
Keep Sorted Elements
stack
queue
priority_queue
Look-up Keys
Insert/erase at middle
Insert/erase at front
Persistent Positions
Frequent Traversals
Size varies Widely
list
vector
deque
Main Purpose
In-order Traversals
Look-up Keys
vector (sorted)
Allow Duplicates
Separate Key / Value
unordered_set
unordered_map
unordered_multiset
unordered_multimap
map
set
multimap
multiset

UML CD containers



Webogaphie

- **C++ Standard Library headers**

https://en.cppreference.com/w/cpp/header

- **Containers**

http://www.cplusplus.com/reference/stl/

- **Introduction to the Standard Template Library**

http://www.martinbroadhurst.com/stl/stl_introduction.html

- **cours STL**

http://www-igm.univ-mlv.fr/~dr/XPOSE2001/stl/Introduction.htm