

La STL

Un concentré de sagesse et d'expérience

Emmanuel Courcelle
V 1.12a



Des conteneurs pour les données

- **templates** = On ne mélange pas les torchons et les serviettes
 - **Allocation mémoire** = Déjà prise en charge
 - **Interface** = Grande cohérence
 - **Itérateurs** = Masquer la complexité interne
 - **Algorithmes** = Fonctions appliquées sur tout le conteneur
-
-

Travaillez en C(++)

- Tableaux extensibles
- Listes liées
- Tableaux associatifs
- chaînes de caractère
- Queues, piles, ...
 - ... avec les performances du C
 - ... sans les complications du C++

Les conteneurs généraux

- Séquentiels
 - Le programmeur décide de l'ordre des éléments
 - *vector, list, deque, queue, stack*
- Ordonnés (associatifs)
 - La bibliothèque décide de l'ordre
 - On peut récupérer les données très rapidement
 - *(multi)map, (multi)set, priority_queue*

Conteneur de quoi ?

D'objets dont le type dispose de:

- ♦ *operator=*
- ♦ *operator==*

Pour un conteneur ordonné, la clé doit avoir:

- ♦ *operator<*



Les conteneurs spécialisés

- **(w)string**
 - Chaînes de caractères ascii ou unicode
 - on oublie strcmp etc.
 - *on n'oublie pas char**
- **Bitset**
 - Tableau de booléens

string

```
string hello = "bonjour ";  
string amis  = "les amis";  
string bye   = "bye bye";  
string P1 = hello + amis;  
string P2 = bye;  
P2 += amis;  
cout << P1 << '\n';  
cout << P2 << '\n';
```

string

- **Ostringstream**
 - Imprimer dans une chaîne de caractères

string

```
cout << hello[3] << '\n';  
int l = hello.length();
```

string: la famille find

```
string::size_type p1 = hello.find('o');  
string::size_type p2 = hello.find('o', p1+1);  
if (hello.find('z') == string::npos)  
    cout << "Pas de z\n";
```

string: substr, erase, insert

```
string h1 = hello.substr(3,4); // jour
        /* (hello n'est pas modifié) */

hello.erase(3,3);           // bonr
        /* (hello est modifié) */

hello.insert(3, "heu");     // bonheur
        /* (hello est modifié) */
```

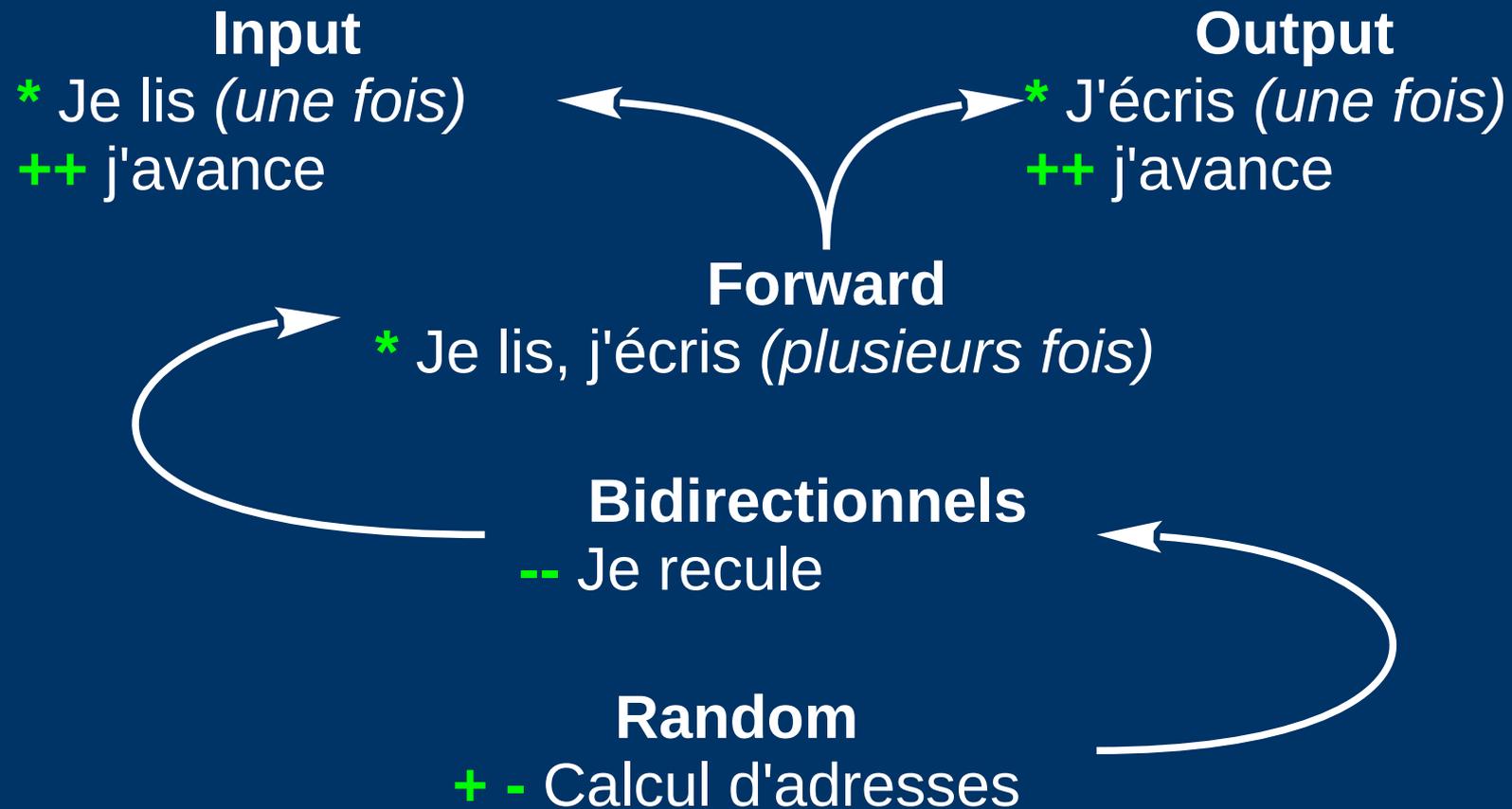
*string et char **

```
string hello = "bonjour";  
string p     =(string)"bonjour " +  
            (string)" les amis";  
  
const char* phrase = p.c_str();  
printf("%s\n", p.c_str());
```

Vecteurs et listes

```
vector<int> V1;  
list<int> L1;  
  
for (int i=0; i<10; i++) {  
    V1.push_back(i);  
    L1.push_back(i);  
}
```

5 types d'itérateurs



Qui peut le plus peut le moins

4 variantes d'itérateurs

```
vector<float> V;
```

```
vector<float>::iterator i;
```

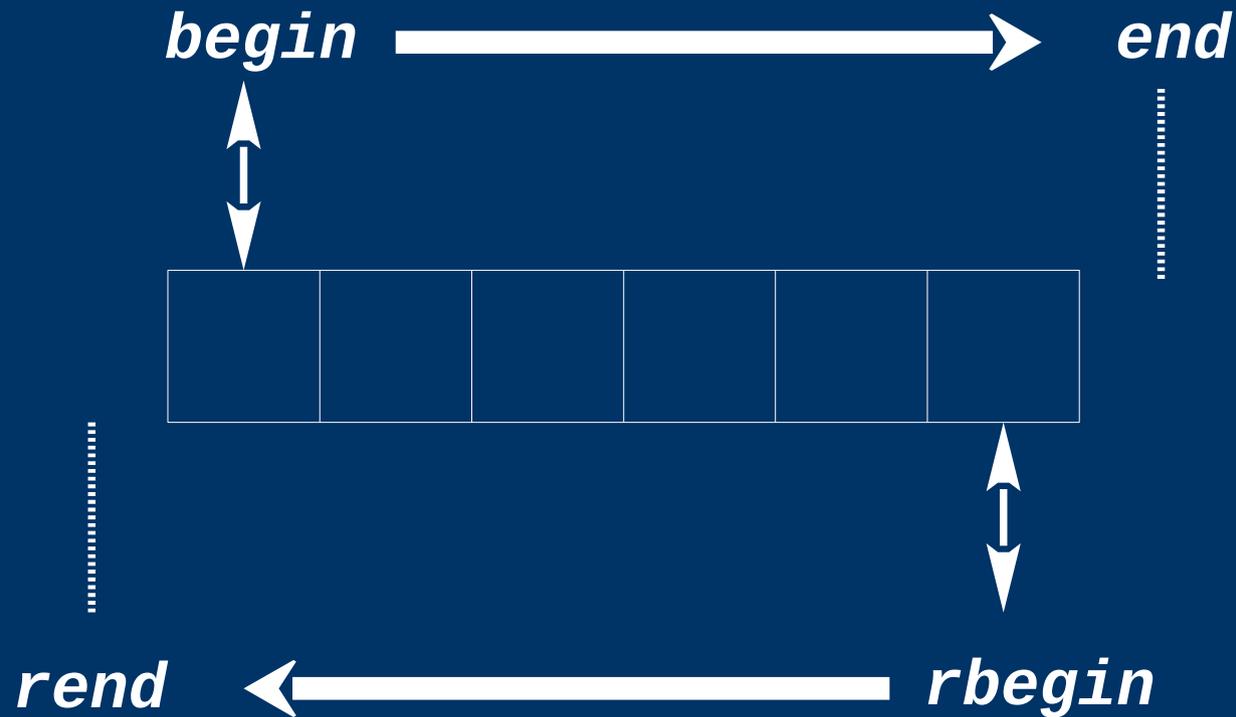
```
vector<float>::const_iterator ci;
```

```
vector<float>::reverse_iterator ri;
```

```
vector<float>::const_reverse_iterator cri;
```



Intervalles: semi-ouverts



[Itérateur1, Itérateur2 [

Balayer un conteneur

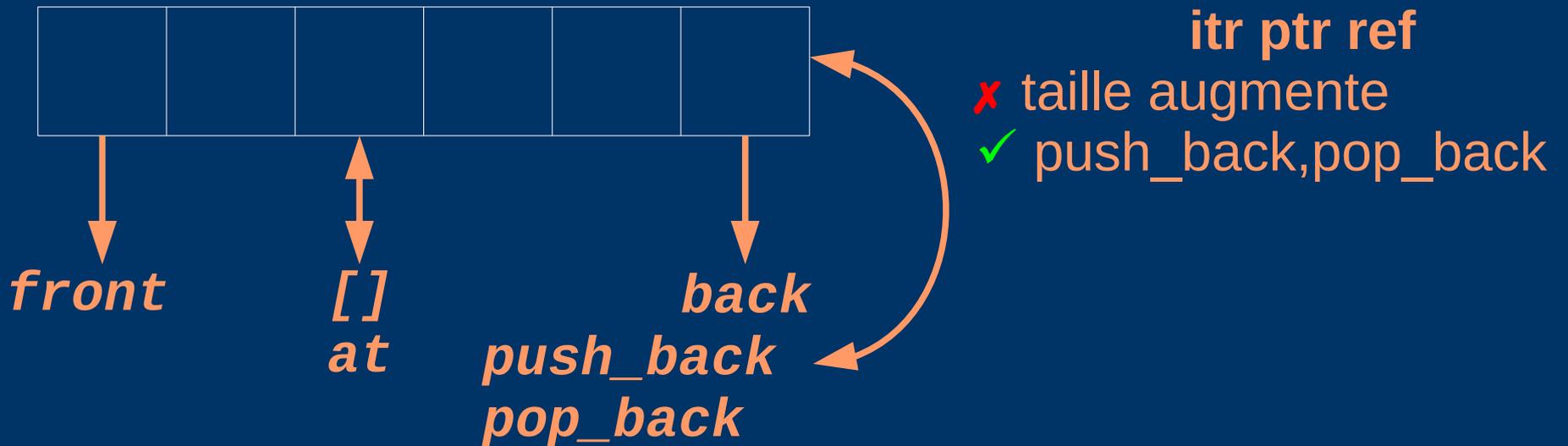
```
conteneur<float> C;  
conteneur<float>::iterator i;  
conteneur<float>::reverse_iterator i;  
for (i=C.begin(); i!=C.end(); ++i){  
    ... *i ...  
}  
for (i=C.rbegin(); i!=C.rend(); ++i){  
    ... *i ...  
}
```

Balayer un conteneur (c++11)

```
conteneur<float> C;  
for (auto i=C.begin(); i!=C.end(); ++i){  
    ... *i ...  
}  
for (auto x : v) {  
    ... x ...  
}  
for (auto i=C.rbegin(); i!=C.rend(); ++i){  
    ... *i ...  
}
```

Vector

Itérateur random



TOUS

clear *size* = ==
erase *max_size* *swap* !=
insert *empty*

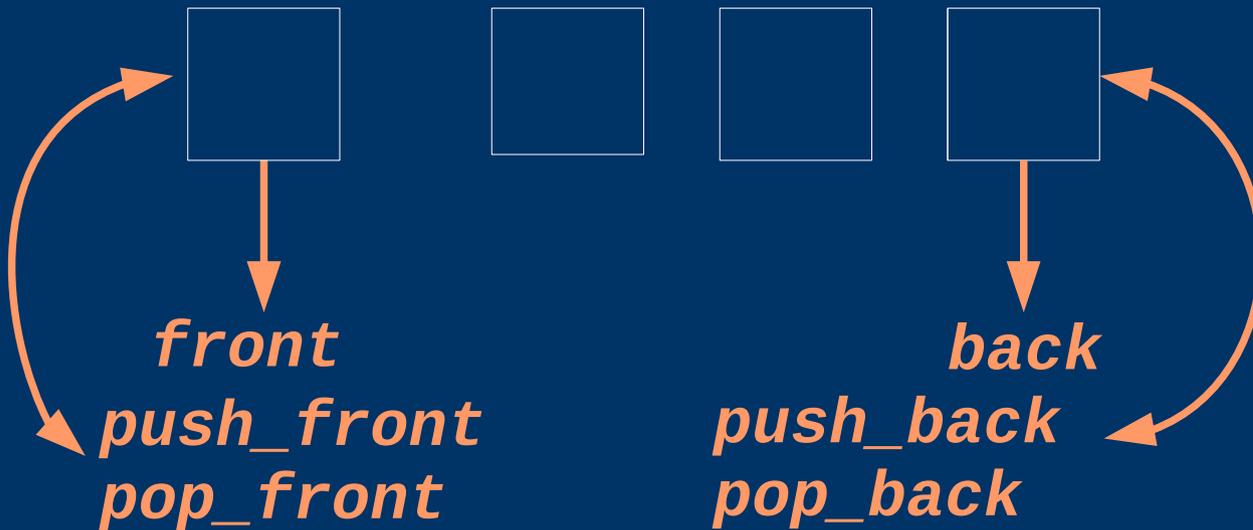
SEQUENTIELS

assign

resize
capacity
reserve

list

itérateur **Bidir**



itr ptr ref
✓ insert, erase
✓ push_x, pop_x

resize
merge
remove
remove_if
reverse
sort
splice
unique

TOUS

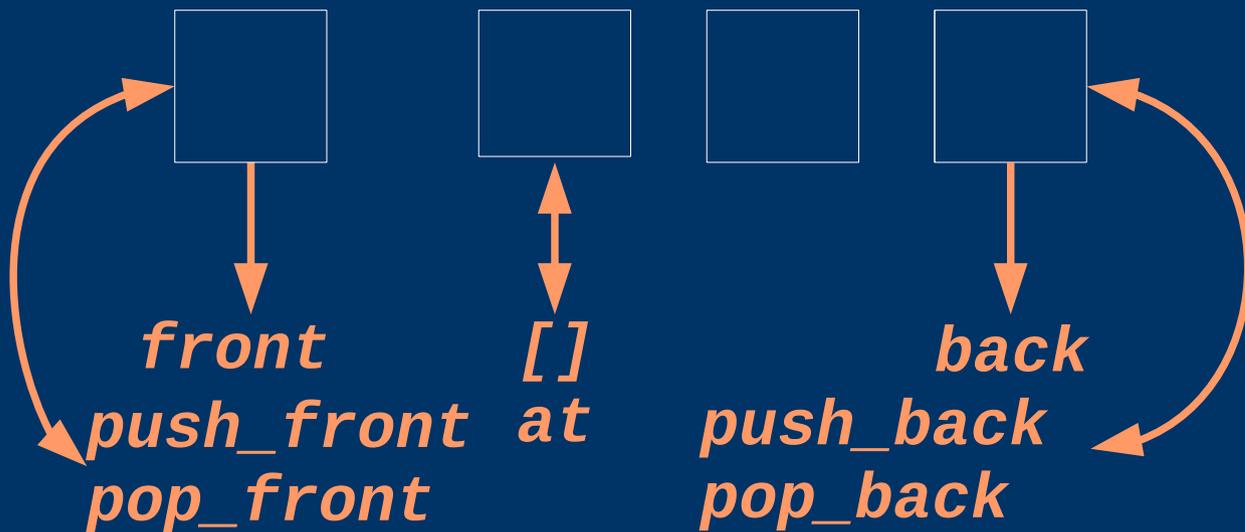
clear *size* = ==
erase *max_size* *swap* !=
insert *empty*

SEQUENTIELS

assign



deque



itérateur random

itr ptr ref
✗ insert, erase

itr
✗ push_x, pop_x

ptr ref
✓ push_x, pop_x

TOUS

clear *size* = ==
erase *max_size* *swap* !=
insert *empty*

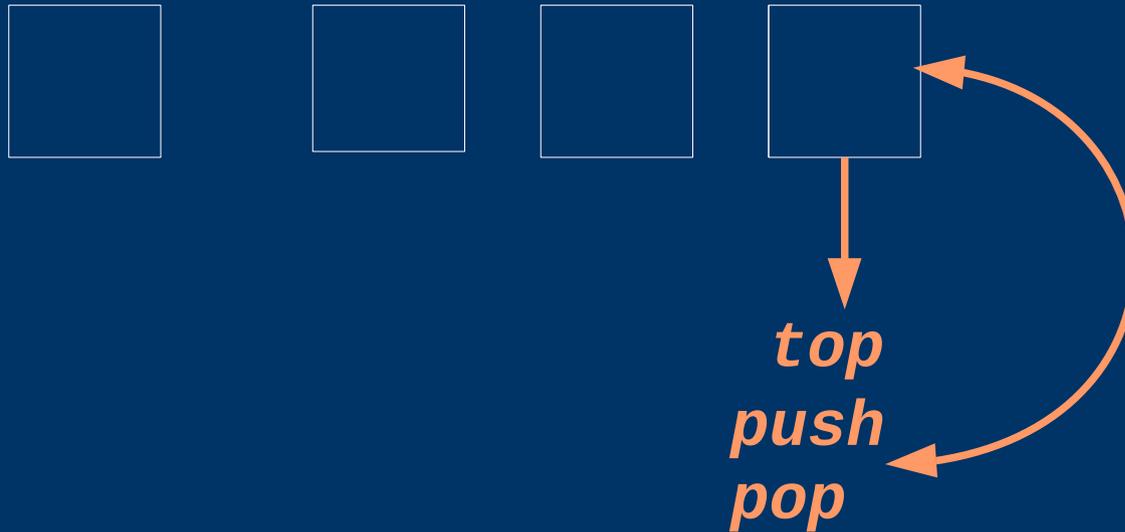
SEQUENTIELS

assign

resize

stack

itérateur **Aucun**



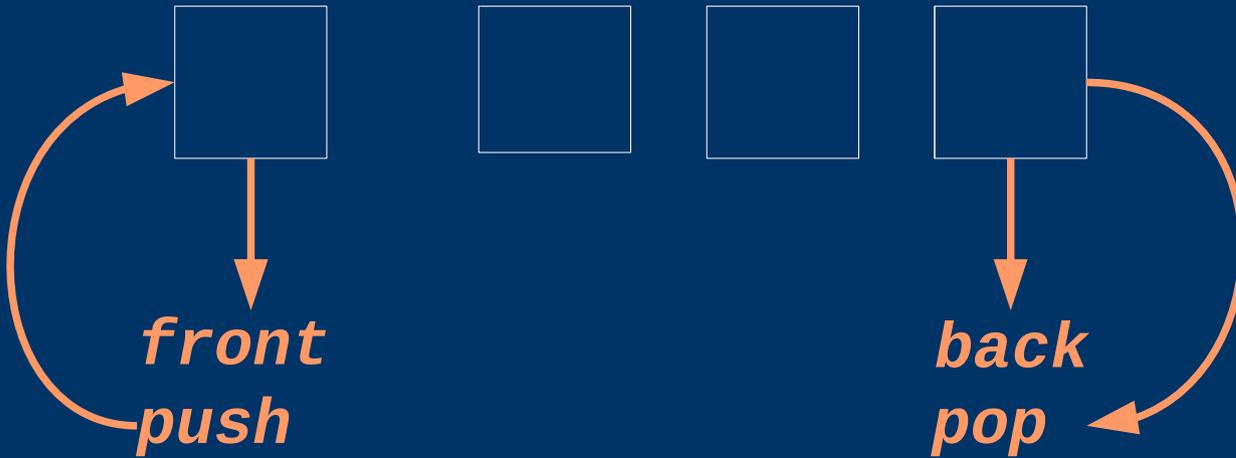
empty
size

Adaptateur

Construit à partir de **deque**, list ou vector

queue

itérateur **Aucun**



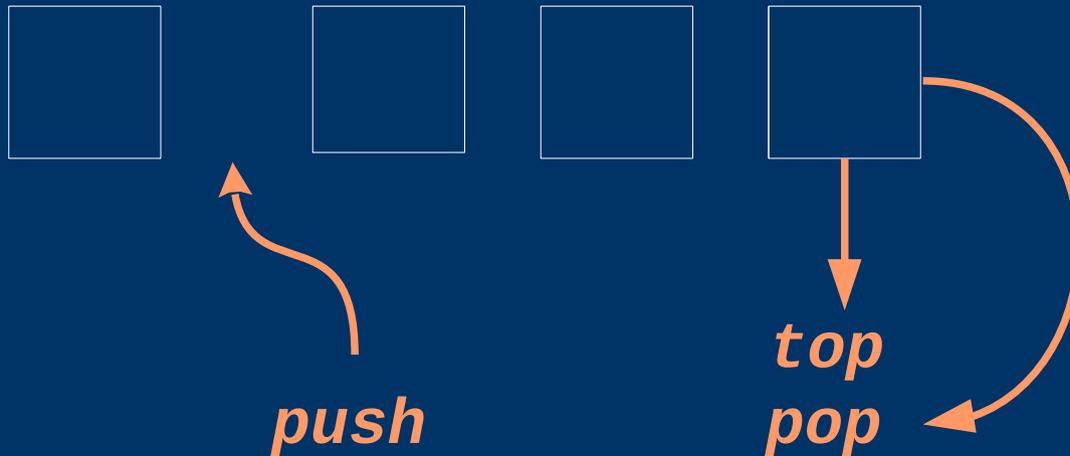
empty
size

Adaptateur

Construit à partir de **deque**, list ou vector

priority_queue

itérateur **Aucun**



Adaptateur ordonné

Construit à partir de **vector**, ou **deque**

empty
size

Attention au pop

```
stack<float> S;  
float t;  
if (!S.empty()) {  
    t = S.top();  
    pop();  
}
```

map, multimap

itérateur **Bidir**
(opère sur les paires)

clés **const**



itr ptr ref

Dans l'ordre croissant des clés ✓ insert, erase

(map seulement) operator[]

Si k existe, renvoie valeur

Sinon, crée une paire (k,v)

TOUS

clear *size* = ==
erase *max_size* *swap* !=
insert *empty*

ORDONNES

find
count
lower_bound
upper_bound

set, multiset

itérateur **Bidir**
(opère sur les paires)
clés **const**



itr ptr ref

Dans l'ordre croissant des clés ✓ insert, erase

TOUS

clear *size* = ==
erase *max_size* *swap* !=
insert *empty*

ORDONNES

find
count
lower_bound
upper_bound

map: examples

```
map<string, string> M1;
map<string, string>::iterator i;

for (i=M1.begin(); i!=M1.end(); ++i){
    cout << "cle= " << i->first << '\n';
    cout << "val= " << i->second << '\n';
}
```

map: examples (c++11)

```
map<string, string> M;
for (auto i=M.begin(); i!=M.end(); ++i){
    cout << "cle= " << i->first << '\n';
    cout << "val= " << i->second << '\n';
}
for (auto x : M){
cout << "cle= " << x.first << '\n';
cout << "val= " << x.second << '\n';
}
```

map: examples

```
map<string, string> M;  
map<string, string>::iterator f;  
string fr = "vert";  
M["blanc"] = "white";  
cout << M["noir"] << '\n';  
f = M.find(fr);  
if (f==M.end())  
    cout << "pas de " << fr << '\n';  
else  
    cout << f->first << " se dit " f->second << '\n';
```

map: exemples (c++11)

```
map<string, string> M;  
string fr = "vert";  
M["blanc"] = "white";  
cout << M["noir"] << '\n';  
auto f = M.find(fr);  
if (f==M.end())  
    cout << "pas de " << fr << '\n';  
else  
    cout << f->first << " se dit " f->second << '\n';
```

Algorithmes

- Utilisent les **itérateurs**
 - Travaillent sur des **intervalles**
 - Fonctionnent avec **tous les conteneurs**
 - Permettent de mettre en relation des conteneurs de **types différents**
-
-

En prévision des exemples

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

/* Quelques conteneurs */
list<float> L,M;
vector<float> V,W;
list::iterator<float> i,j;

/* Un prédicat */
bool pos(float x) { return x>=0?true:false;}

/* Une fonction génératrice */
float gen() { return time()/1000;}
```

Généralités

```
int count(Inp first, Inp last, constT& val);  
int count_if(Inp first, Inp last, Pred pred);  
Function for_each(Inp first, Inp last, Func func);  
  
cout << count_if(L.begin(),L.end(),pos) << '\n';
```

Comparaison

```
bool equal(Inp1 first1, Inp1 last1, Inp2 first2);  
bool equal(Inp1 first1, Inp1 last1, Inp2 first2, Pred pred);  
bool lexicographical_compare(Inp1 first1, Inp1 last1, Inp2  
    first2, Inp2 last2);  
min, max, min_element, max_element
```

```
if (equal(L.begin(), L.begin()+10, V.begin(), pos)) ...  
if (lexicographical_compare(L.begin(), L.end(),  
    V.begin(), V.end())) ...
```

Recherche, remplacement

```
Fwd adjacent_find(Fwd first, Fwd last)
```

```
Inp find(Ind first, Inp last, const T& value)
```

```
find_end, find_first_of, find_if
```

```
Fwd1 search(Fwd1 first, Fwd1 last, Fwd2 first2, Fd2 last2)
```

```
void replace(Fwd first, Fwd last, const T& old, const T& new)
```

```
if (find(V.begin(), V.end(), 0) == V.end())  
    cout << "aucun element egal a 0\n";
```

Copie, suppression

Out copy(Inp first, Int last, Out result)

↳ Attention à l'allocation mémoire

```
void fill(Fwd first, Fwd last, const T& val);
```

```
void generate(Fwd first, Fwd last, Gen gen);
```

```
Fwd remove_if (Fwd first, Fwd last, Out result,  
              Pred pred);
```

```
V.erase(remove_if(L.begin(), L.end(), i, pos), last);
```

Réarrangements

```
void random_shuffle(Rnd first, Rnd last);  
void reverse(Bidi first, bidi last);  
void rotate(Fwd first, Fwd middle, Fwd last);  
Fwd2 swap_ranges(Fwd first1, Fwd last1, Fwd2 first2);
```

⚠ Attention à l'allocation mémoire

```
// 0,1,2,3,4,5,6,7,8,9 => 1,2,3,4,0,5,6,7,8,9  
rotate(V.begin(), V.begin()+1, V.begin()+5)
```