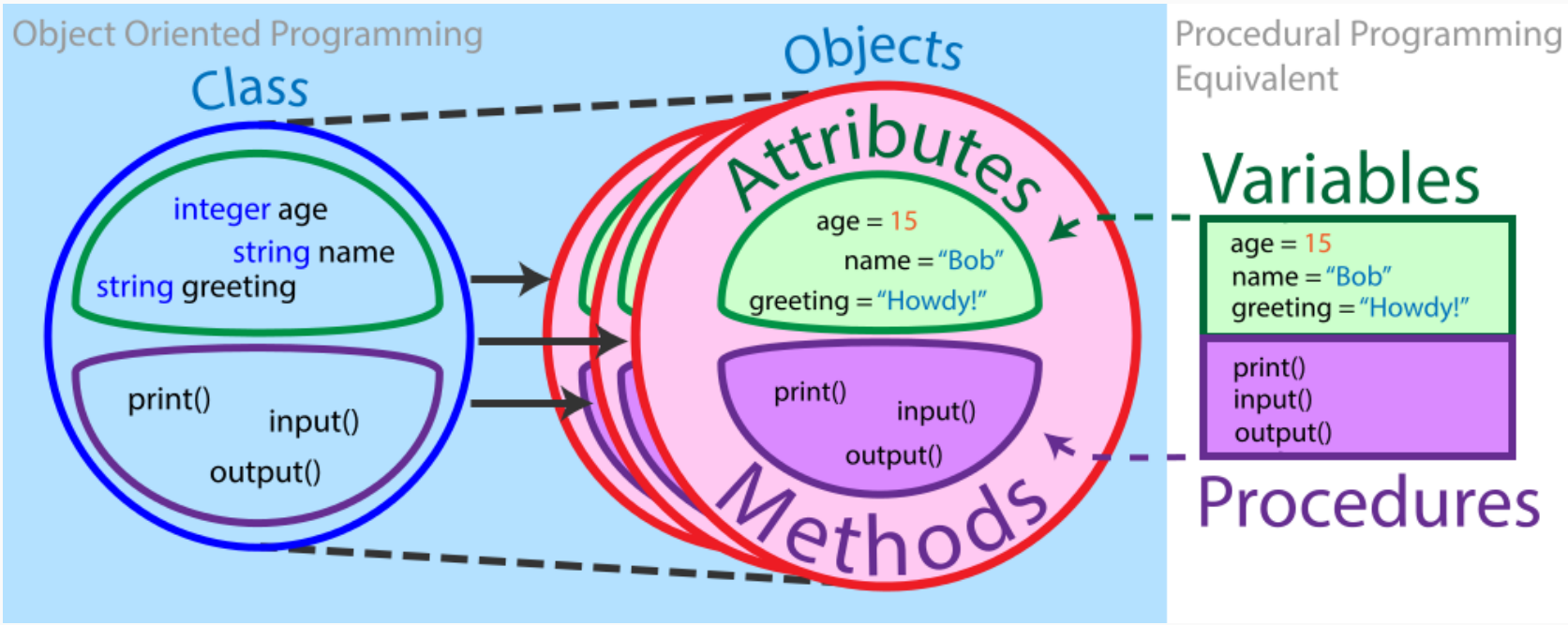
Programmation Orientée Objet — Episode #2

Retour sur les classes

Schématiquement



Généralités

```
1  #ifndef POINT_HPP
2  #define POINT_HPP
3
4  #include <iostream>
5
6  class Point {
7
8    private: // par défaut --- TBC
9      int x;
10     int y;
11   public:
12     Point(); // aucun type de retour
13     Point(const Point &p);
14
15     // Doivent être const... Suffisant ?
16     int get_x() const { return x; }
17     int get_y() const { return y; }
18
```

Généralités

```cpp
#include "Point.hpp"

using namespace std;

Point::Point() {
  cout << "Constructeur par défaut." << endl;
  (*this).x = 0;
  (*this).y = 0;
}

Point::Point(const Point &p) {
  cout << "Constructeur par recopie." << endl;
  (*this).x = p.get_x();
  (*this).y = p.get_y();
}
```

Généralités

```
1  #include <iostream>
2  #include "Point.hpp"
3
4  static Point comparer(Point p, Point p1) {
5    return p.get_x() < p1.get_x() ? p : p1;
6  }
7
8  int main() {
9    Point p;
10   Point p1;
11   comparer(p, p1);
12
13   return 0;
14 }
```

Tableaux améliorés

Les tableaux en C++

Fonctionnalités manquantes ?

Implémentation d'une classe `Tableau`

La librairie standard — STL

La STL contient quatre composantes principales :

- Conteneurs
- Itérateurs
- Algorithmes
- Fonctions

Standard Template Library

Ensemble de classes de structures de données :

- vecteurs
- listes
- files, piles
- ...

d'algorithmes :

- tri
- recherche
- modification
- ...

et bien plus !

Les conteneurs

Le ”conteneur” string — TBC

```cpp
#include <iostream>
#include <string>
#include <typeinfo>

using namespace std;

int main() {

  string s = "Ceci n'est pas une chaine de caracteres";
  cout << s << endl;

  char* str = (char*)s.c_str();
  cout << str << " " << *str << endl; // ?

  return 0;

}
```

Le ”conteneur” string — TBC

```cpp
#include <iostream>
#include <string>
#include <typeinfo>

using namespace std;

int main() {

  string s = "Ceci n'est pas une chaine de caracteres";
  std::cout << typeid(s).name() << '\n';
  cout << s << endl;

  char* str = (char*)s.c_str();
  cout << str << " " << *str << " " << static_cast<void*>(str)
       << endl;

  return 0;

}
```

Le ”conteneur” string — TBC

```cpp
#include <iostream>
#include <string>
#include <typeinfo>

using namespace std;

int main() {

  string s1 = "Ceci est ";
  string s2 = "une concatenation";
  string s3 = s1 + s2;

  cout << s1 + s2 << endl;

  return 0;

}
```

Le ”conteneur” `string` — TBC

Member functions

(constructor)	constructs a `basic_string` (public member function)
(destructor)	destroys the string, deallocating internal storage if used (public member function)
`operator=`	assigns values to the string (public member function)
`assign`	assign characters to a string (public member function)
`get_allocator`	returns the associated allocator (public member function)
Element access	
`at`	accesses the specified character with bounds checking (public member function)
`operator[]`	accesses the specified character (public member function)
`front` (C++11)	accesses the first character (public member function)
`back` (C++11)	accesses the last character (public member function)
`data`	returns a pointer to the first character of a string (public member function)
`c_str`	returns a non-modifiable standard C character array version of the string (public member function)
`operator basic_string_view` (C++17)	returns a non-modifiable string_view into the entire string (public member function)
Iterators	
`begin` **`cbegin`** (C++11)	returns an iterator to the beginning (public member function)
`end` **`cend`** (C++11)	returns an iterator to the end (public member function)
`rbegin` **`crbegin`** (C++11)	returns a reverse iterator to the beginning (public member function)
`rend` **`crend`** (C++11)	returns a reverse iterator to the end (public member function)
Capacity	
`empty`	checks whether the string is empty (public member function)
`size` **`length`**	returns the number of characters (public member function)
`max_size`	returns the maximum number of characters (public member function)
`reserve`	reserves storage

Le ”conteneur” `string` — TBC

Operations	
`clear`	clears the contents (public member function)
`insert`	inserts characters (public member function)
`erase`	removes characters (public member function)
`push_back`	appends a character to the end (public member function)
`pop_back` (C++11)	removes the last character (public member function)
`append`	appends characters to the end (public member function)
`operator+=`	appends characters to the end (public member function)
`compare`	compares two strings (public member function)
`starts_with` (C++20)	checks if the string starts with the given prefix (public member function)
`ends_with` (C++20)	checks if the string ends with the given suffix (public member function)
`replace`	replaces specified portion of a string (public member function)
`substr`	returns a substring (public member function)
`copy`	copies characters (public member function)
`resize`	changes the number of characters stored (public member function)
`swap`	swaps the contents (public member function)
Search	
`find`	find characters in the string (public member function)
`rfind`	find the last occurrence of a substring (public member function)
`find_first_of`	find first occurrence of characters (public member function)
`find_first_not_of`	find first absence of characters (public member function)
`find_last_of`	find last occurrence of characters

Conteneurs séquentiels

`vector` : structure de données proche des tableaux.

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

int main() {

  std::vector<int> v(6, 1); // v: 1,1,1,1,1,1
  std::vector<int> v1(v); // v1: 1,1,1,1,1,1

  std::vector<int> v2 = v;
  std::cout << &v << " " << &v2 << std::endl; // ?
  v[3] = 2; // v: 1,1,1,2,1,1
  v1[1] = 0; // v: 1,1,1,2,1,1

  return 0;

}
```

Conteneurs séquentiels

Et en mémoire ?

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

int main() {

  std::vector<int> v(6, 1);

  for(int i = 0; i < v.size(); i++)
    std::cout << &v[i] << " ";
  std::cout << std::endl;

  return 0;

}
```

Tableaux dynamiques avec gestion automatique de la mémoire — Zone mémoire **contigüe**.

Conteneurs séquentiels

```
1  static void tailles(const std::vector<int>& nums1,
2              const std::vector<int>& nums2,
3              const std::vector<int>& nums3) {
4    std::cout << "nums1: " << nums1.size()
5          << " nums2: " << nums2.size()
6          << " nums3: " << nums3.size() << '\n';
7  }
```

Conteneurs séquentiels

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

int main() {
  std::vector<int> nums1(10, 1);
  std::vector<int> nums2;
  std::vector<int> nums3;

  tailles(nums1, nums2, nums3);

  // *copie*
  nums2 = nums1;
  tailles(nums1, nums2, nums3);

  return 0;
}
```

Conteneurs séquentiels

En détails

Member functions

(constructor)	constructs the `vector` (public member function)
(destructor)	destructs the `vector` (public member function)
`operator=`	assigns values to the container (public member function)
`assign`	assigns values to the container (public member function)
`get_allocator`	returns the associated allocator (public member function)

Element access

`at`	access specified element with bounds checking (public member function)
`operator[]`	access specified element (public member function)
`front`	access the first element (public member function)
`back`	access the last element (public member function)
`data` (C++11)	direct access to the underlying array (public member function)

Iterators

`begin` `cbegin`	returns an iterator to the beginning (public member function)
`end` `cend`	returns an iterator to the end (public member function)
`rbegin` `crbegin`	returns a reverse iterator to the beginning (public member function)
`rend` `crend`	returns a reverse iterator to the end (public member function)

Capacity

`empty`	checks whether the container is empty (public member function)
`size`	returns the number of elements (public member function)
`max_size`	returns the maximum possible number of elements (public member function)
`reserve`	reserves storage (public member function)
`capacity`	returns the number of elements that can be held in currently allocated storage (public member function)
`shrink_to_fit` (C++11)	reduces memory usage by freeing unused memory (public member function)

Modifiers

`clear`	clears the contents (public member function)
`insert`	inserts elements (public member function)
`emplace` (C++11)	constructs element in-place (public member function)
`erase`	erases elements (public member function)
`push_back`	adds an element to the end (public member function)
`emplace_back` (C++11)	constructs an element in-place at the end (public member function)
`pop_back`	removes the last element (public member function)
`resize`	changes the number of elements stored (public member function)
`swap`	swaps the contents (public member function)

Conteneurs séquentiels

Et aussi :

`list`	mémoire non-contigüe
`deque`	similaire à `vector` avec gestion des deux côtés.
	contigüité en mémoire non garantie
`queue`	file
`priority_queue`	file de priorité
`stack`	pile

Conteneurs associatifs

Structures de données *ordonnées* pouvant être parcourues *rapidement*.

Principalement :

`(multi)set`	ensemble d'éléments uniques (valeur $==$ clé)
	plusieurs éléments de même valeur
`(multi)map`	association clé-valeur
	plusieurs éléments de même clé

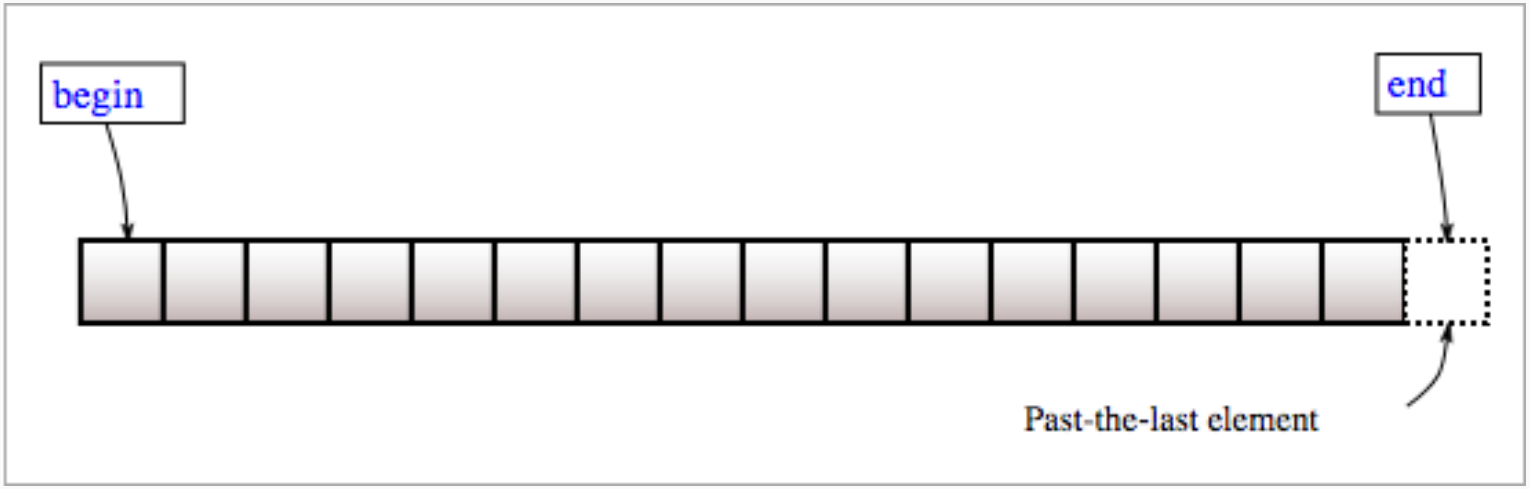
En pratique ?

- Critère d'ordre ?
- Représentation ?
- **Parcours ?**
- ...

Les itérateurs

Les fonctions `begin()` et `end()`

Syntaxe : `conteneur::iterator`



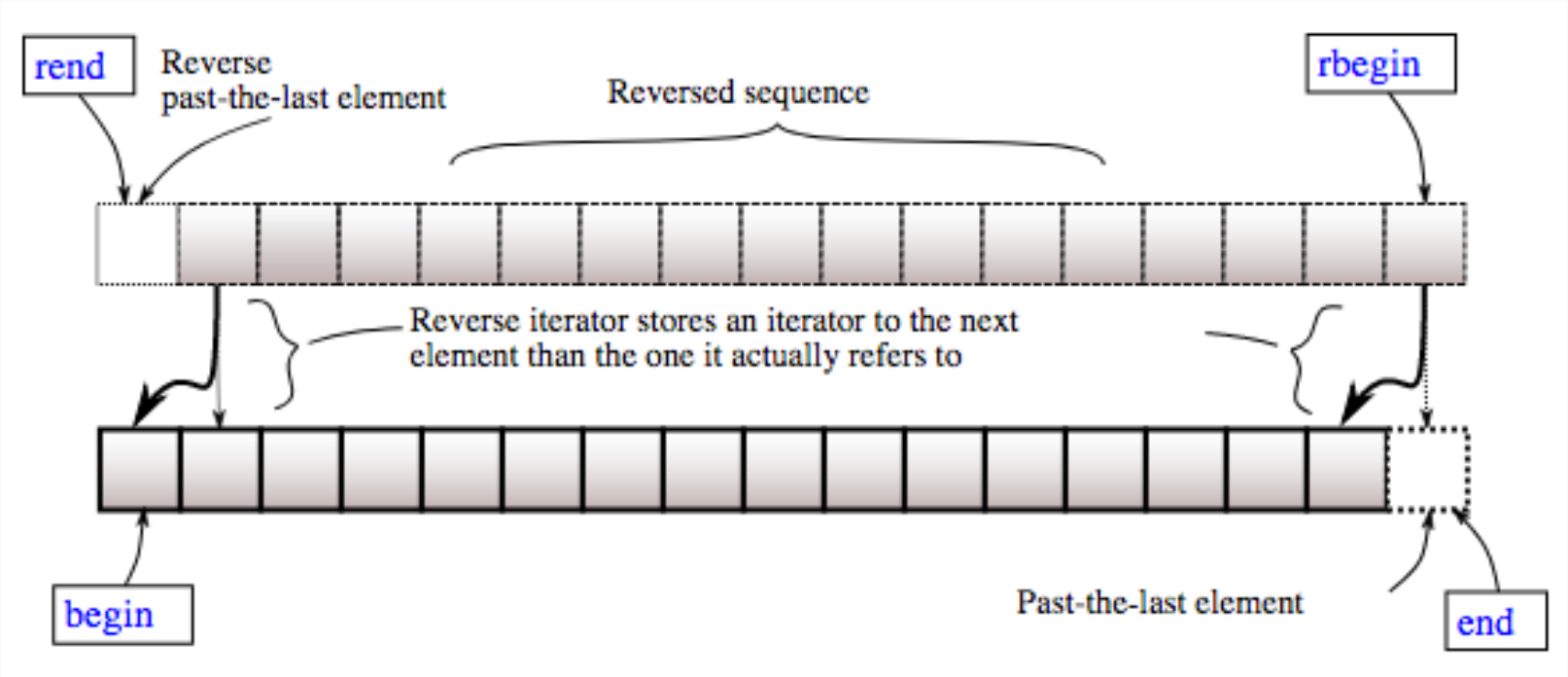
Les fonctions begin() et end()

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

int main() {

  std::vector<int> v(6, 1);
  std::vector<int>::iterator it;

  for(it = v.begin(); it < v.end(); it++)
    std::cout << (*it) << ",";

  std::cout << *(v.end()) << std::endl;

  return 0;

}
```

Les fonctions begin() et end()

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

int main() {

  std::vector<int> v(6, 1);
  std::vector<int>::iterator it;

  for(it = v.begin(); it < v.end(); it++)
    std::cout << &(*it) << " " << *it << " ";

  std::cout << std::endl <<
  &(*(v.end())) << "," << *(v.end()) << std::endl;

  // ?

  return 0;

}
```

Les fonctions `rbegin()` et `rend()`



Un peu d'arithmétique

Similaire à l'arithmétique des pointeurs... ... pour les `RandomAccessIterator`.

Penser : conteneurs séquentiels.

La recherche dans un conteneur — `find`

La fonction `find` retourne un itérateur sur l'élément trouvé.

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

int main() {

  std::vector<int> v(6, 1);
  std::vector<int>::iterator it;

  it = find(v.begin(), v.end(), 1);
  if(it != v.end()) std::cout << "Trouvé !" << std::endl;

  it = find(v.begin(), v.end(), 2);
  if(it == v.end()) std::cout << "Non trouvé !" << std::endl;

  return 0;

}
```

Retour aux conteneurs associatifs — map

```
1  static map<char, int> count_char(fstream &input) {
2    map<char, int> m;
3    char c;
4    while(input >> noskipws >> c)
5      ++m[c];
6
7    return m;
8  }
```

Retour aux conteneurs associatifs — map

```cpp
#include <iostream>
#include <fstream>
#include <map>

using namespace std;

int main() {
  fstream input("input.in");
  map<char, int> m = count_char(input);

  map<char, int>::iterator it;
  for(it = m.begin(); it != m.end(); it++)
    if((*it).first != '\n')
      cout << (*it).first << " " << (*it).second << endl;

  input.close();

  return 0;
}
```

Retour aux conteneurs associatifs

En détails

Member functions

(constructor)	constructs the map (public member function)
(destructor)	destructs the map (public member function)
`operator=`	assigns values to the container (public member function)
`get_allocator`	returns the associated allocator (public member function)

Element access

`at` (C++11)	access specified element with bounds checking (public member function)
`operator[]`	access or insert specified element (public member function)

Iterators

`begin` `cbegin`	returns an iterator to the beginning (public member function)
`end` `cend`	returns an iterator to the end (public member function)
`rbegin` `crbegin`	returns a reverse iterator to the beginning (public member function)
`rend` `crend`	returns a reverse iterator to the end (public member function)

Capacity

`empty`	checks whether the container is empty (public member function)
`size`	returns the number of elements (public member function)
`max_size`	returns the maximum possible number of elements (public member function)

Modifiers

`clear`	clears the contents (public member function)
`insert`	inserts elements or nodes (since C++17) (public member function)
`insert_or_assign` (C++17)	inserts an element or assigns to the current element if the key already exists (public member function)
`emplace` (C++11)	constructs element in-place (public member function)
`emplace_hint` (C++11)	constructs elements in-place using a hint (public member function)
`try_emplace` (C++17)	inserts in-place if the key does not exist, does nothing if the key exists (public member function)
`erase`	erases elements (public member function)
`swap`	swaps the contents (public member function)
`extract` (C++17)	extracts nodes from the container (public member function)
`merge` (C++17)	splices nodes from another container (public member function)

Lookup

`count`	returns the number of elements matching specific key (public member function)
`find`	finds element with specific key (public member function)
`contains` (C++20)	checks if the container contains element with specific key (public member function)
`equal_range`	returns range of elements matching a specific key (public member function)
`lower_bound`	returns an iterator to the first element *not less* than the given key (public member function)
`upper_bound`	returns an iterator to the first element *greater* than the given key (public member function)

Observers

`key_comp`	returns the function that compares keys (public member function)
`value_comp`	returns the function that compares keys in objects of type value_type (public member function)

Itérateurs et conteneurs

Category	Container	After **insertion**, are...		After **erasure**, are...		Conditionally
		iterators valid?	**references** valid?	**iterators** valid?	**references** valid?	
Sequence containers	`array`	N/A		N/A		
	`vector`	No		N/A		Insertion changed capacity
		Yes		Yes		Before modified element(s)
		No		No		At or after modified element(s)
	`deque`	No	Yes	Yes, except erased element(s)		Modified first or last element
			No	No		Modified middle only
	`list`	Yes		Yes, except erased element(s)		
	`forward_list`	Yes		Yes, except erased element(s)		
Associative containers	`set` `multiset` `map` `multimap`	Yes		Yes, except erased element(s)		
Unordered associative containers	`unordered_set` `unordered_multiset`	No	Yes	N/A		Insertion caused rehash
	`unordered_map` `unordered_multimap`	Yes		Yes, except erased element(s)		No rehash

Les algorithmes

Quelques exemples

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

int main() {
  std::vector<int> v;

  for(int i = 10; i > 0; i--) v.push_back(i);
  std::vector<int> v1 = v;

  std::sort(v.begin(), v.end());

  for(int i = 0; i < 10; i++) std::cout << v[i] << " ";
  std::cout << std::endl;

  return 0;
}
```

Quelques exemples

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

int main() {
  std::vector<int> v;
  for(int i = 10; i > 0; i--) v.push_back(i);
  std::vector<int> v1 = v;

  std::sort(v.begin(), v.end());

  for(int i = 0; i < 10; i++) std::cout << v[i] << " ";
  std::cout << std::endl;

  std::sort(v1.begin()+3, v1.begin()+7);
  for(int i = 0; i < 10; i++) std::cout << v1[i] << " ";

  return 0;
}
```

Comment trier dans l'ordre décroissant ?

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

int main() {
  std::vector<int> v;
  for(int i = 0; i < 10; i++) v.push_back(i);
  std::vector<int> v1 = v;

  // ordre décroissant
  std::sort(v.rbegin(), v.rend());
  for(int i = 0; i < 10; i++) std::cout << v[i] << " ";
  std::cout << std::endl;
  // ordre pré-défini
  std::sort(v1.begin(), v1.end(), std::greater<int>());
  for(int i = 0; i < 10; i++) std::cout << v1[i] << " ";

  return 0;
}
```

L'implémentation

Les vector

Member types

Member type	Definition
value_type	T
allocator_type	Allocator
size_type	Unsigned integer type (usually std::size_t)
difference_type	Signed integer type (usually std::ptrdiff_t)
reference	Allocator::reference (until C++11) value_type& (since C++11)
const_reference	Allocator::const_reference (until C++11) const value_type& (since C++11)
pointer	Allocator::pointer (until C++11) std::allocator_traits<Allocator>::pointer (since C++11)
const_pointer	Allocator::const_pointer (until C++11) std::allocator_traits<Allocator>::const_pointer (since C++11)
iterator	*RandomAccessIterator*
const_iterator	Constant *RandomAccessIterator*
reverse_iterator	std::reverse_iterator<iterator>
const_reverse_iterator	std::reverse_iterator<const_iterator>

Les set

Structure **ordonnée** sans duplicité.

Member types

Member type	Definition
key_type	Key
value_type	Key
size_type	Unsigned integer type (usually std::size_t)
difference_type	Signed integer type (usually std::ptrdiff_t)
key_compare	Compare
value_compare	Compare
allocator_type	Allocator
reference	Allocator::reference (until C++11) value_type& (since C++11)
const_reference	Allocator::const_reference (until C++11) const value_type& (since C++11)
pointer	Allocator::pointer (until C++11) std::allocator_traits<Allocator>::pointer (since C++11)
const_pointer	Allocator::const_pointer (until C++11) std::allocator_traits<Allocator>::const_pointer (since C++11)
iterator	Constant *BidirectionalIterator*
const_iterator	Constant *BidirectionalIterator*
reverse_iterator	std::reverse_iterator<iterator>
const_reverse_iterator	std::reverse_iterator<const_iterator>
node_type(since C++17)	a specialization of node handle representing a container node

Les algorithmes

Toujours se poser la question de l'implémentation.

L'algorithme `sort` :

- tri par sélection : $O(n^2)$
- tri bulle : $O(n^2)$
- tri fusion : $O(n \log n)$
- **quicksort :** $O(n \log n)$

Une liste complète est disponible ici.