STL
(Standard Template Library)

By Payongkit XI

WHAT IS STL?

The Standard Template Library (STL) คือคลาสชนิดหนึ่งใน c++ ที่รวบรวมทั้งโครงสร้างข้อมูลและอัลกอริทึมพื้นฐานเข้าไว้ด้วยกัน เช่น vector, lists, stacks, Queue, Sort algo, etc.

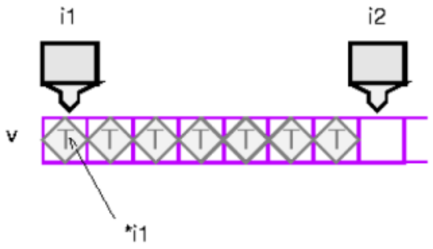
ซึ่งหลักๆแล้ว ก็แบ่งได้เป็น 3 ส่วน

- Containers (เก็บข้อมูล)
- Iterators (เข้าถึงข้อมูล)
- Algorithms (จัดการข้อมูล)

Iterators

Iterators represent locations in a container.
Each container has its own iterator type.

```
vector<int> v;
// add some integers to v
vector::iterator i1 = v.begin();
vector::iterator i2 = v.end()
```



will create two iterators like this picture:

Iterators behave like regular pointers ...

- <, >
- ++, --
- ==, !=

But work for all the containers!

- container.begin()

// return iterator of first elem.

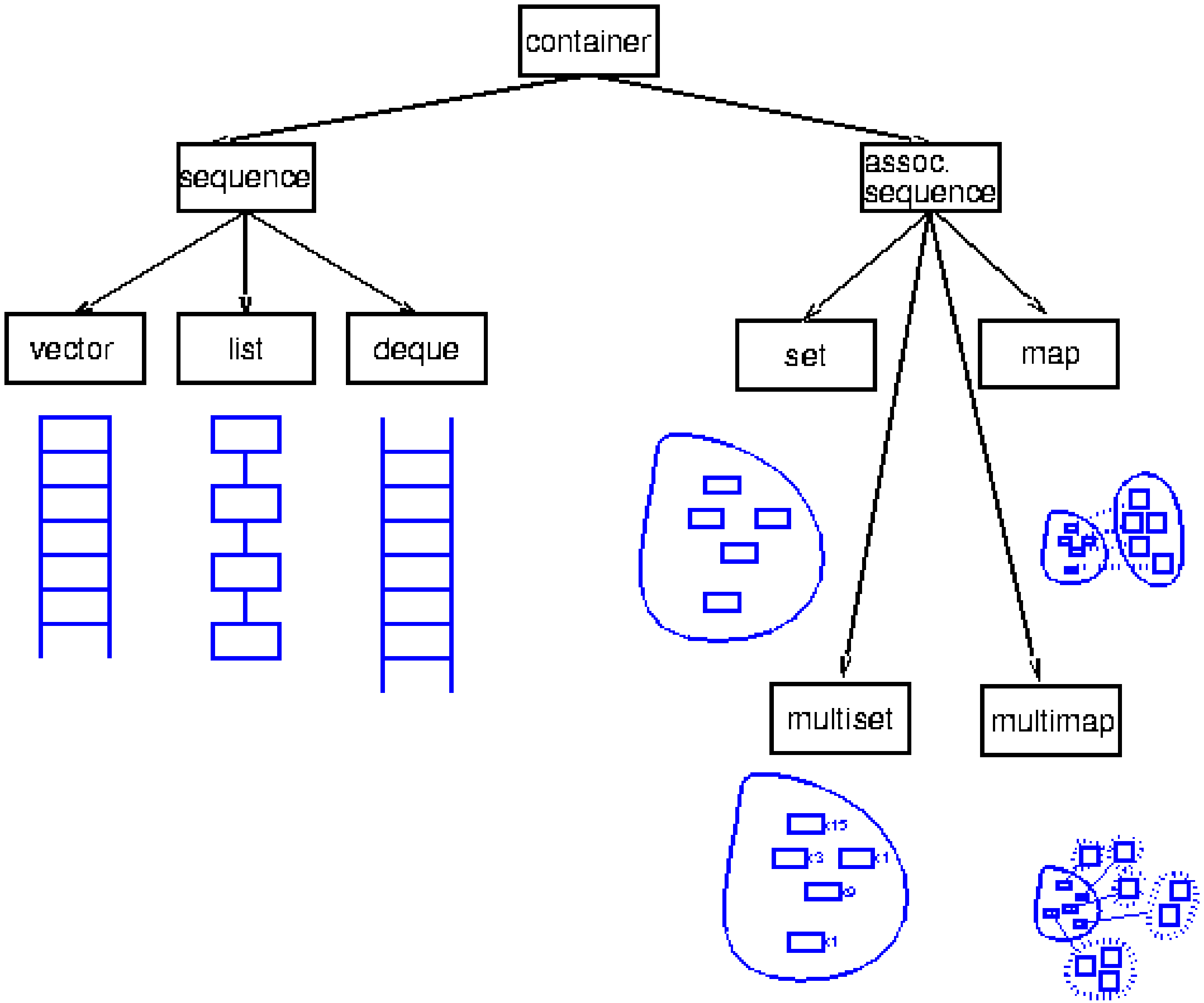
- container.end()

// return iterator next to last elem.

Containers

- Sequence Containers: implement data structures which can be accessed in a sequential manner.
  - vector
  - list
  - deque
  - arrays
  - forward_list (Introduced in C++11)
- Container Adaptors : provide a different interface for sequential containers.
  - queue
  - priority_queue
  - stack
- Associative Containers : implement sorted data structures that can be quickly searched (O(log n) complexity).
  - set
  - multiset
  - map
  - multimap
- Unordered Associative Containers : implement unordered data structures that can be quickly searched
  - unordered_set (Introduced in C++11)
  - unordered_multiset (Introduced in C++11)
  - unordered_map (Introduced in C++11)
  - unordered_multimap (Introduced in C++11)

Containers


container
sequence
assoc. sequence
vector
list
deque
set
map
multiset
multimap
x15
x3
x1
x9
x1

Containers : Vector

Containers : Vector

Header

#include <vector>

Constructors

vector<T> v;	Make an empty vector.	O(1)
vector<T> v(n);	Make a vector with N elements.	O(n)
vector<T> v(n, value);	Make a vector with N elements, initialized to value.	O(n)
vector<T> v(begin, end);	Make a vector and copy the elements from begin to end.	O(n)

Accessors

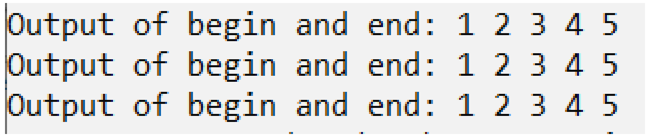
v[i];	Return (or set) the I'th element.	O(1)
v.at(i);	Return (or set) the I'th element, with bounds checking.	O(1)
v.size();	Return current number of elements.	O(1)
v.empty();	Return true if vector is empty.	O(1)
v.begin();	Return random access iterator to start.	O(1)
v.end();	Return random access iterator to end.	O(1)
v.front();	Return the first element.	O(1)
v.back();	Return the last element.	O(1)
v.capacity();	Return maximum number of elements.	O(1)

Modifiers

v.push_back(value);	Add value to end.	O(1)
v.insert(iterator, value);	Insert value at the position indexed by iterator.	O(n)
v.pop_back();	Remove value from end.	O(1)
v.erase(iterator);	Erase value indexed by iterator.	O(n)
v.erase(begin, end);	Erase the elements from begin to end.	
v.resize(n)	Resizes the container so that it contains ‘n’ elements.	
v.shrink_to_fit()	Reduces the capacity of the container to fit its size and destroys all elements beyond the capacity.	
v.emplace(iterator, value)	Like insert but is preferred for efficiency reasons with object.	O(n)
v.emplace_back(value)	Like push_back but is preferred for efficiency reasons with object.	O(1)

Containers : Vector

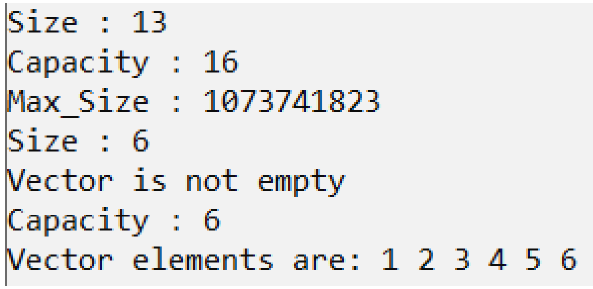
```
#include <iostream>
#include <vector>
#include <iterator>
using namespace std;
int main()
{
    vector<int> g1;

    vector<int>  :: iterator it;

    for (int i = 1; i <= 5; i++)
        g1.push_back(i);

    cout << "Output of begin and end: ";
    for (int i = 0; i < g1.size(); i++)
        cout << g1[i] << " ";

    cout << "\nOutput of begin and end: ";
    for (it = g1.begin(); it != g1.end(); it++)
        cout << *it << " ";

    cout << "\nOutput of begin and end: ";
    for (auto i = g1.begin(); i != g1.end(); i++)
        cout << *i << " ";

    return 0;
}
```



```
Output of begin and end: 1 2 3 4 5
Output of begin and end: 1 2 3 4 5
Output of begin and end: 1 2 3 4 5
```

Containers : Vector

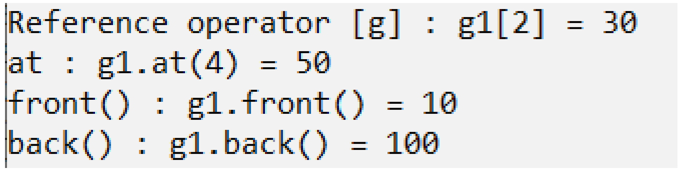
```
#include <iostream>
#include <vector>
#include <iterator>
using namespace std;
int main()
{
    vector<int> g1;
    for (int i = 1; i <= 13; i++)
        g1.push_back(i);

    cout << "Size : " << g1.size();
    cout << "\nCapacity : " << g1.capacity();
    cout << "\nMax_Size : " << g1.max_size();

    // resizes the vector size
    g1.resize(6);
    // prints the vector size after resize()
    cout << "\nSize : " << g1.size();

    // checks if the vector is empty or not
    if (g1.empty() == false)
        cout << "\nVector is not empty";
    else
        cout << "\nVector is empty";

    // Shrinks the vector
    g1.shrink_to_fit();
    cout << "\nCapacity : " << g1.capacity();

    cout << "\nVector elements are: ";
    for (auto it = g1.begin(); it != g1.end(); it++)
        cout << *it << " ";

    return 0;
}
```



```
Size : 13
Capacity : 16
Max_Size : 1073741823
Size : 6
Vector is not empty
Capacity : 6
Vector elements are: 1 2 3 4 5 6
```

Containers : Vector

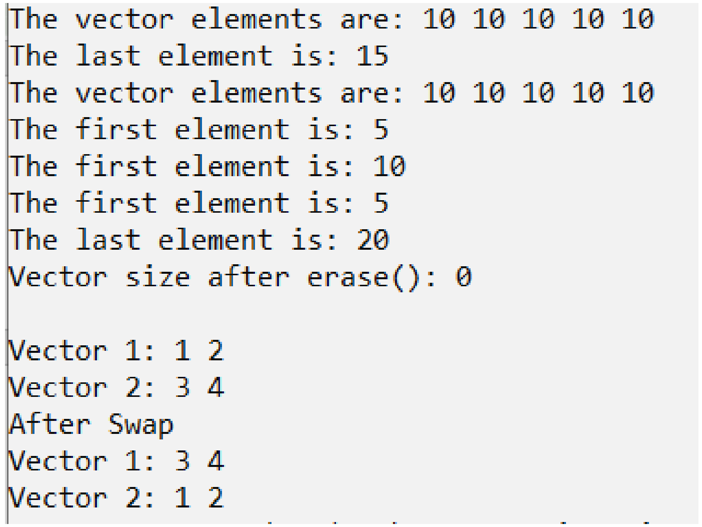
```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> g1;

    for (int i = 1; i <= 10; i++)
        g1.push_back(i * 10);

    cout << "\nReference operator [g] : g1[2] = " << g1[2];

    cout << "\nat : g1.at(4) = " << g1.at(4);

    cout << "\nfront() : g1.front() = " << g1.front();

    cout << "\nback() : g1.back() = " << g1.back();

    return 0;
}
```



```
Reference operator [g] : g1[2] = 30
at : g1.at(4) = 50
front() : g1.front() = 10
back() : g1.back() = 100
```

Containers : Vector

```cpp
#include <bits/stdc++.h>
using namespace std;

main()
{
    // Assign vector
    vector<int> v;

    // fill the array with 10 five times
    v.assign(5, 10);

    cout << "The vector elements are: ";
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";

    // inserts 15 to the last position
    v.push_back(15);
    int n = v.size();
    cout << "\nThe last element is: " << v[n - 1];

    // removes last element
    v.pop_back();

    // prints the vector
    cout << "\nThe vector elements are: ";
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " ";


    // inserts 5 at the beginning
    v.insert(v.begin(), 5);

    cout << "\nThe first element is: " << v[0];


    // removes the first element
    v.erase(v.begin());

    cout << "\nThe first element is: " << v[0];

    // inserts at the beginning
    v.emplace(v.begin(), 5);
    cout << "\nThe first element is: " << v[0];

    // Inserts 20 at the end
    v.emplace_back(20);
    n = v.size();
    cout << "\nThe last element is: " << v[n - 1];

    // erases the vector
    v.clear();
    cout << "\nVector size after erase(): " << v.size();

    // two vector to perform swap
    vector<int> v1, v2;
    v1.push_back(1);
    v1.push_back(2);
    v2.push_back(3);
    v2.push_back(4);

    cout << "\n\nVector 1: ";
    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << " ";

    cout << "\nVector 2: ";
    for (int i = 0; i < v2.size(); i++)
        cout << v2[i] << " ";

    // Swaps v1 and v2
    v1.swap(v2);

    cout << "\nAfter Swap \nVector 1: ";
    for (int i = 0; i < v1.size(); i++)
        cout << v1[i] << " ";

    cout << "\nVector 2: ";
    for (int i = 0; i < v2.size(); i++)
        cout << v2[i] << " ";
}
```



```
The vector elements are: 10 10 10 10 10
The last element is: 15
The vector elements are: 10 10 10 10 10
The first element is: 5
The first element is: 10
The first element is: 5
The last element is: 20
Vector size after erase(): 0

Vector 1: 1 2
Vector 2: 3 4
After Swap
Vector 1: 3 4
Vector 2: 1 2
```

Containers : List

Containers : List

Header
#include <list>

Constructors

list<T> l;	Make an empty list.	O(1)
list<T> l(begin, end);	Make a list and copy the values from begin to end.	O(n)

Accessors

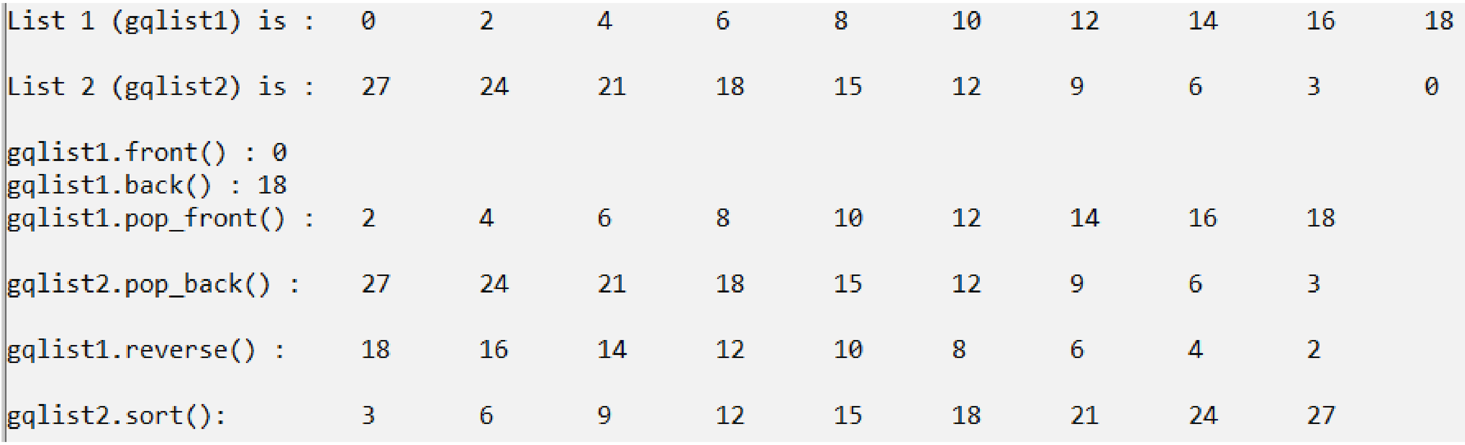
l.size();	Return current number of elements.	O(1)
l.empty();	Return true if list is empty.	O(1)
l.begin();	Return bidirectional iterator to start.	O(1)
l.end();	Return bidirectional iterator to end.	O(1)
l.front();	Return the first element.	O(1)
l.back();	Return the last element.	O(1)

Modifiers

l.push_front(value);	Add value to front.	O(1)
l.push_back(value);	Add value to end.	O(1)
l.insert(iterator, value);	Insert value after position indexed by iterator.	O(1)
l.pop_front();	Remove value from front.	O(1)
l.pop_back();	Remove value from end.	O(1)
l.erase(iterator);	Erase value indexed by iterator.	O(1)
l.erase(begin, end);	Erase the elements from begin to end.	O(1)
l.remove(value);	Remove all occurrences of value.	O(n)
l.remove_if(test);	Remove all element that satisfy test.	O(n)
l.reverse();	Reverse the list.	O(n)
l.sort();	Sort the list.	O(n log n)
l.sort(comparison);	Sort with comparison function.	O(n logn)
l.merge(l2);	Merge sorted lists.	O(n)

Containers : List

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

//function for printing the elements in a list
void showlist(list <int> g)
{
    list <int> :: iterator it;
    for(it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}

int main()
{

    list <int> gqlist1, gqlist2;


    for (int i = 0; i < 10; ++i)
    {
        gqlist1.push_back(i * 2);
        gqlist2.push_front(i * 3);
    }
    cout << "\nList 1 (gqlist1) is : ";
    showlist(gqlist1);

    cout << "\nList 2 (gqlist2) is : ";
    showlist(gqlist2);

    cout << "\ngqlist1.front() : " << gqlist1.front();
    cout << "\ngqlist1.back() : " << gqlist1.back();

    cout << "\ngqlist1.pop_front() : ";
    gqlist1.pop_front();
    showlist(gqlist1);

    cout << "\ngqlist2.pop_back() : ";
    gqlist2.pop_back();
    showlist(gqlist2);

    cout << "\ngqlist1.reverse() : ";
    gqlist1.reverse();
    showlist(gqlist1);

    cout << "\ngqlist2.sort(): ";
    gqlist2.sort();
    showlist(gqlist2);

    return 0;

}
```



```
List 1 (gqlist1) is :   0       2       4       6       8       10      12      14      16      18

List 2 (gqlist2) is :   27      24      21      18      15      12      9       6       3       0

gqlist1.front() : 0
gqlist1.back() : 18
gqlist1.pop_front() :   2       4       6       8       10      12      14      16      18

gqlist2.pop_back() :    27      24      21      18      15      12      9       6       3

gqlist1.reverse() :     18      16      14      12      10      8       6       4       2

gqlist2.sort():         3       6       9       12      15      18      21      24      27
```

Containers : Deque

Containers : Deque

Header

#include <deque>

Constructors

deque<T> d;	Make an empty deque.	O(1)
deque<T> d(n);	Make a deque with N elements.	O(n)
deque<T> d(n, value);	Make a deque with N elements, initialized to value.	O(n)
deque<T> d(begin, end);	Make a deque and copy the values from begin to end.	O(n)

Accessors

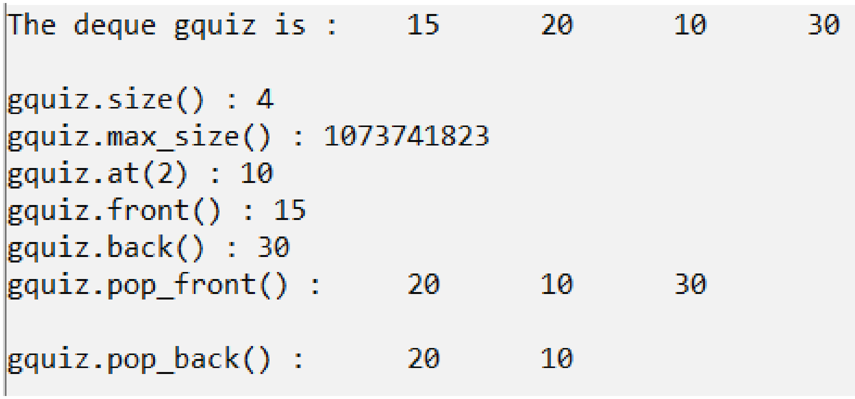
d[i];	Return (or set) the I'th element.	O(1)
d.at(i);	Return (or set) the I'th element, with bounds checking.	O(1)
d.size();	Return current number of elements.	O(1)
d.empty();	Return true if deque is empty.	O(1)
d.begin();	Return random access iterator to start.	O(1)
d.end();	Return random access iterator to end.	O(1)
d.front();	Return the first element.	O(1)
d.back();	Return the last element.	O(1)

Modifiers

d.push_front(value);	Add value to front.	O(1)
d.push_back(value);	Add value to end.	O(1)
d.insert(iterator, value);	Insert value at the position indexed by iterator.	O(n)
d.pop_front();	Remove value from front.	O(1)
d.pop_back();	Remove value from end.	O(1)
d.erase(iterator);	Erase value indexed by iterator.	O(n)
d.erase(begin, end);	Erase the elements from begin to end.	O(n)

Containers : Deque

```cpp
#include <iostream>
#include <deque>
using namespace std;
void showdq(deque <int> g)
{
    deque <int> :: iterator it;
    for (it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}
int main()
{
    deque <int> gquiz;
    gquiz.push_back(10);
    gquiz.push_front(20);
    gquiz.push_back(30);
    gquiz.push_front(15);
    cout << "The deque gquiz is : ";
    showdq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.max_size() : " << gquiz.max_size();

    cout << "\ngquiz.at(2) : " << gquiz.at(2);
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop_front() : ";
    gquiz.pop_front();
    showdq(gquiz);

    cout << "\ngquiz.pop_back() : ";
    gquiz.pop_back();
    showdq(gquiz);

    return 0;
}
```



```
The deque gquiz is :    15      20      10      30

gquiz.size() : 4
gquiz.max_size() : 1073741823
gquiz.at(2) : 10
gquiz.front() : 15
gquiz.back() : 30
gquiz.pop_front() :     20      10      30

gquiz.pop_back() :      20      10
```

Containers : Stack

Containers : Stack

Header
#include <stack>

Constructors

stack<T> s;	Make an empty stack.	O(1)
stack< container<T> > s;	Make an empty stack.	O(1)

Accessors

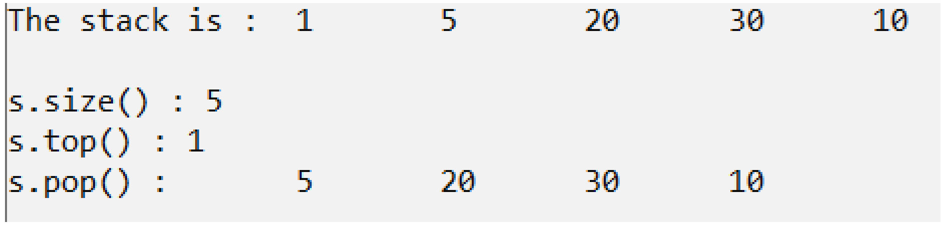
s.top();	Return the top element.	O(1)
s.size();	Return current number of elements.	O(1)
s.empty();	Return true if stack is empty.	O(1)

Modifiers

s.push(value);	Push value on top.	Same as push_back() for underlying container.
s.pop();	Pop value from top.	O(1)

Containers : Stack

```
#include <bits/stdc++.h>
using namespace std;

void showstack(stack <int> s)
{
    while (!s.empty())
    {
        cout << '\t' << s.top();
        s.pop();
    }
    cout << '\n';
}

int main ()
{
    stack <int> s;
    s.push(10);
    s.push(30);
    s.push(20);
    s.push(5);
    s.push(1);

    cout << "The stack is : ";
    showstack(s);

    cout << "\ns.size() : " << s.size();
    cout << "\ns.top() : " << s.top();


    cout << "\ns.pop() : ";
    s.pop();
    showstack(s);

    return 0;
}
```



```
The stack is :  1       5       20      30      10

s.size() : 5
s.top() : 1
s.pop() :       5       20      30      10
```

Containers : Queue

Containers : Queue

Header
#include <queue>

Constructors

queue<T> q;	Make an empty queue.	O(1)
queue< container<T> > q;	Make an empty queue.	O(1)

Accessors

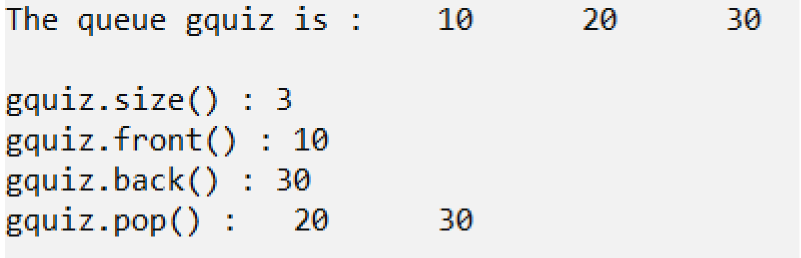
q.front();	Return the front element.	O(1)
q.back();	Return the rear element.	O(1)
q.size();	Return current number of elements.	O(1)
q.empty();	Return true if queue is empty.	O(1)

Modifiers

q.push(value);	Add value to end.	Same for push_back() for underlying container.
q.pop();	Remove value from front.	O(1)

Containers : Queue

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

void showq(queue <int> gq)
{
    queue <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.front();
        g.pop();
    }
    cout << '\n';
}

int main()
{
    queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);

    cout << "The queue gquiz is : ";
    showq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);

    return 0;
}
```



```
The queue gquiz is :    10      20      30

gquiz.size() : 3
gquiz.front() : 10
gquiz.back() : 30
gquiz.pop() :   20      30
```

Containers : Priority Queue

Containers : Priority Queue

Header
#include <queue>

Constructors

priority_queue<T, container<T>, comparison<T> > q;	Make an empty priority queue using the given container to hold values, and comparison to compare values. container defaults to vector<T> and comparison defaults to less<T>.	O(1)

Accessors

q.top();	Return the "biggest" element.	O(1)
q.size();	Return current number of elements.	O(1)
q.empty();	Return true if priority queue is empty.	O(1)

Modifiers

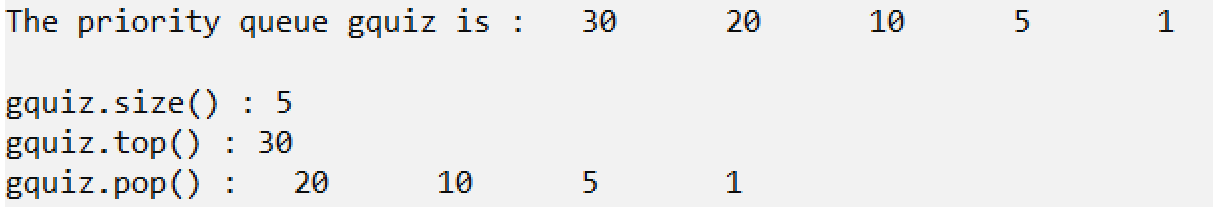
q.push(value);	Add value to priority queue.	O(log n)
q.pop();	Remove biggest value.	O(log n)

```
// Syntax to create a max heap for priority queue
priority_queue <int, vector<int>, less<int>> g = gq;
```

```
// Syntax to create a min heap for priority queue
priority_queue <int, vector<int>, greater<int>> g = gq;
```

Containers : Priority Queue

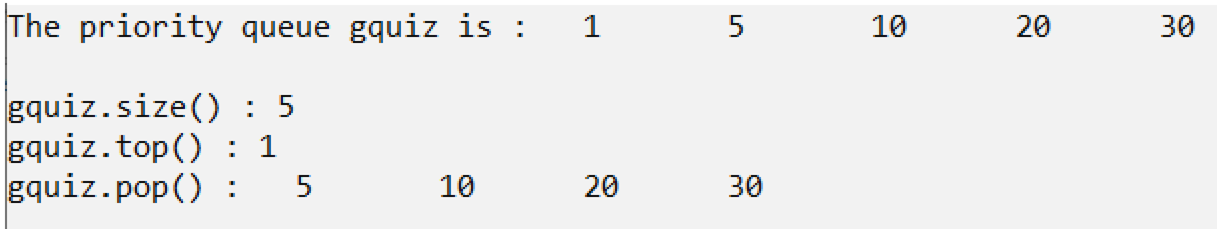
```cpp
// Note that by default C++ creates a max-heapfor priority queue
#include <iostream>
#include <queue>
using namespace std;

void showpq(priority_queue <int> gq)
{
    priority_queue <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}
int main ()
{
    priority_queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The priority queue gquiz is : ";
    showpq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();

    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);

    return 0;
}
```



```
The priority queue gquiz is :   30      20      10      5       1

gquiz.size() : 5
gquiz.top() : 30
gquiz.pop() :   20      10      5       1
```

Containers : Priority Queue

```
// C++ program to demonstrate min heap
#include <iostream>
#include <queue>
using namespace std;

void showpq(priority_queue <int, vector<int>, greater<int>> gq)
{
    priority_queue <int, vector<int>, greater<int>> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}
int main ()
{
    priority_queue <int, vector<int>, greater<int>> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The priority queue gquiz is : ";
    showpq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();


    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);

    return 0;
}
```



```
The priority queue gquiz is :   1       5       10      20      30

gquiz.size() : 5
gquiz.top() : 1
gquiz.pop() :   5       10      20      30
```

Containers : Set and Multiset

Containers : Set and Multiset

Header

#include <set>

Constructors

set< type, compare > s;	Make an empty set. compare should be a binary predicate for ordering the set. It's optional and will default to a function that uses operator<.	O(1)
set< type, compare > s(begin, end);	Make a set and copy the values from begin to end.	O(n log n)

Accessors

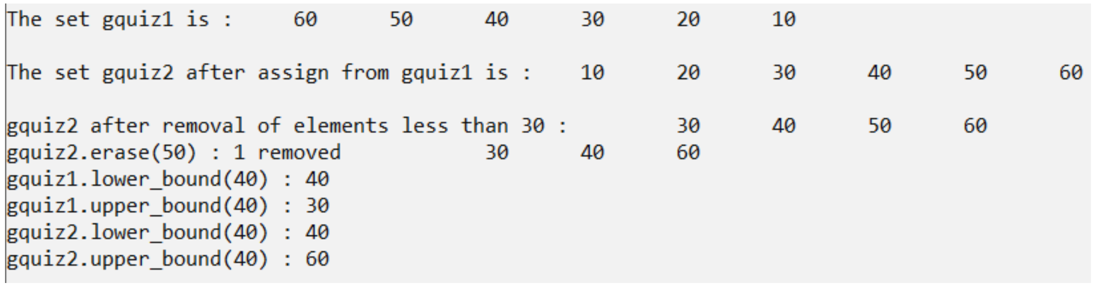
s.find(key)	Return an iterator pointing to an occurrence of key in s, or s.end() if key is not in s.	O(log n)
s.lower_bound(key)	Return an iterator pointing to the first occurrence of an item in s not less than key, or s.end() if no such item is found.	O(log n)
s.upper_bound(key)	Return an iterator pointing to the first occurrence of an item greater than key in s, or s.end() if no such item is found.	O(log n)
s.equal_range(key)	Returns pair<lower_bound(key), upper_bound(key)>.	O(log n)
s.count(key)	Returns the number of items equal to key in s.	O(log n)
s.size();	Return current number of elements.	O(1)
s.empty();	Return true if set is empty.	O(1)
s.begin()	Return an iterator pointing to the first element.	O(1)
s.end()	Return an iterator pointing one past the last element.	O(1)

Modifiers

s.insert(iterator, key)	Inserts key into s. iterator is taken as a "hint" but key will go in the correct position no matter what. Returns an iterator pointing to where key went.	O(log n)
s.insert(key)	Inserts key into s and returns a pair<iterator, bool>, where iterator is where key went and bool is true if key was actually inserted, i.e., was not already in the set.	O(log n)

Containers : Set and Multiset

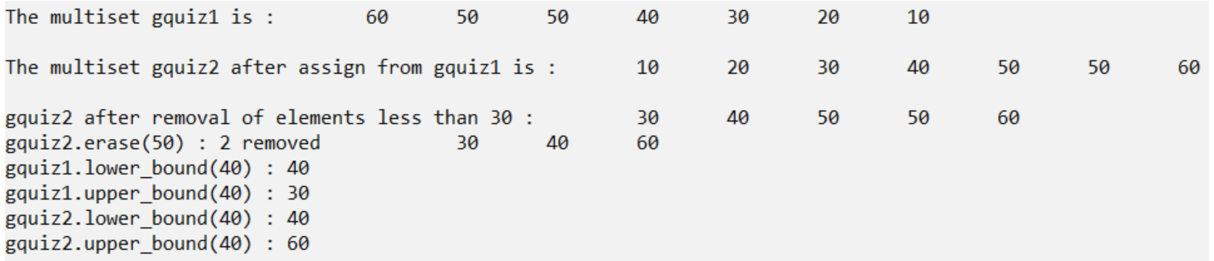
```cpp
#include <iostream>
#include <set>
#include <iterator>
using namespace std;
int main()
{
    // empty set container
    set <int, greater <int> > gquiz1;

    // insert elements in random order
    gquiz1.insert(40);
    gquiz1.insert(30);
    gquiz1.insert(60);
    gquiz1.insert(20);
    gquiz1.insert(50);
    gquiz1.insert(50); // only one 50 will be added to the set
    gquiz1.insert(10);

    // printing set gquiz1
    set <int, greater <int> > :: iterator itr;
    cout << "\nThe set gquiz1 is : ";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;

    // assigning the elements from gquiz1 to gquiz2
    set <int> gquiz2(gquiz1.begin(), gquiz1.end());

    // print all elements of the set gquiz2
    cout << "\nThe set gquiz2 after assign from gquiz1 is : ";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;

    // remove all elements up to 30 in gquiz2
    cout << "\ngquiz2 after removal of elements less than 30 : ";
    gquiz2.erase(gquiz2.begin(), gquiz2.find(30));
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << *itr;
    }

    // remove element with value 50 in gquiz2
    int num;
    num = gquiz2.erase (50);
    cout << "\ngquiz2.erase(50) : ";
    cout << num << " removed \t" ;
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;
    //lower bound and upper bound for set gquiz1
    cout << "gquiz1.lower_bound(40) : "
         << *gquiz1.lower_bound(40) << endl;
    cout << "gquiz1.upper_bound(40) : "
         << *gquiz1.upper_bound(40) << endl;

    //lower bound and upper bound for set gquiz2
    cout << "gquiz2.lower_bound(40) : "
         << *gquiz2.lower_bound(40) << endl;
    cout << "gquiz2.upper_bound(40) : "
         << *gquiz2.upper_bound(40) << endl;

    return 0;

}
```



```
The set gquiz1 is :     60      50      40      30      20      10

The set gquiz2 after assign from gquiz1 is :    10      20      30      40      50      60

gquiz2 after removal of elements less than 30 :         30      40      50      60
gquiz2.erase(50) : 1 removed            30      40      60
gquiz1.lower_bound(40) : 40
gquiz1.upper_bound(40) : 30
gquiz2.lower_bound(40) : 40
gquiz2.upper_bound(40) : 60
```

Containers : Set and Multiset

```cpp
#include <iostream>
#include <set>
#include <iterator>
using namespace std;
int main()
{
    // empty multiset container
    multiset <int, greater <int> > gquiz1;
    // insert elements in random order
    gquiz1.insert(40);
    gquiz1.insert(30);
    gquiz1.insert(60);
    gquiz1.insert(20);
    gquiz1.insert(50);
    gquiz1.insert(50); // 50 will be added again to the multiset unlike set
    gquiz1.insert(10);

    // printing multiset gquiz1
    multiset <int, greater <int> > :: iterator itr;
    cout << "\nThe multiset gquiz1 is : ";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;

    // assigning the elements from gquiz1 to gquiz2
    multiset <int> gquiz2(gquiz1.begin(), gquiz1.end());

    // print all elements of the multiset gquiz2
    cout << "\nThe multiset gquiz2 after assign from gquiz1 is : ";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;
    // remove all elements up to element with value 30 in gquiz2
    cout << "\ngquiz2 after removal of elements less than 30 : ";
    gquiz2.erase(gquiz2.begin(), gquiz2.find(30));
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    // remove all elements with value 50 in gquiz2
    int num;
    num = gquiz2.erase(50);
    cout << "\ngquiz2.erase(50) : ";
    cout << num << " removed \t" ;
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << *itr;
    }
    cout << endl;
    //lower bound and upper bound for multiset gquiz1
    cout << "gquiz1.lower_bound(40) : "
         << *gquiz1.lower_bound(40) << endl;
    cout << "gquiz1.upper_bound(40) : "
         << *gquiz1.upper_bound(40) << endl;
    //lower bound and upper bound for multiset gquiz2
    cout << "gquiz2.lower_bound(40) : "
         << *gquiz2.lower_bound(40) << endl;
    cout << "gquiz2.upper_bound(40) : "
         << *gquiz2.upper_bound(40) << endl;

         return 0;

}
```



```
The multiset gquiz1 is :        60      50      50      40      30      20      10

The multiset gquiz2 after assign from gquiz1 is :       10      20      30      40      50      50      60

gquiz2 after removal of elements less than 30 :         30      40      50      50      60
gquiz2.erase(50) : 2 removed            30      40      60
gquiz1.lower_bound(40) : 40
gquiz1.upper_bound(40) : 30
gquiz2.lower_bound(40) : 40
gquiz2.upper_bound(40) : 60
```

Containers : Map and Multimap

Containers : Map and Multimap

Header

#include <map>

Constructors

map< key_type, value_type, key_compare > m;	Make an empty map. key_compare should be a binary predicate for ordering the keys. It's optional and will default to a function that uses operator<.	O(1)
map< key_type, value_type, key_compare > m(begin, end);	Make a map and copy the values from begin to end.	O(n log n)

Accessors

m[key]	Return the value stored for key. This adds a default value if key not in map.	O(log n)
m.find(key)	Return an iterator pointing to a key-value pair, or m.end() if key is not in map.	O(log n)
m.lower_bound(key)	Return an iterator pointing to the first pair containing key, or m.end() if key is not in map.	O(log n)
m.upper_bound(key)	Return an iterator pointing one past the last pair containing key, or m.end() if key is not in map.	O(log n)
m.equal_range(key)	Return a pair containing the lower and upper bounds for key. This may be more efficient than calling those functions separately.	O(log n)
m.size();	Return current number of elements.	O(1)
m.empty();	Return true if map is empty.	O(1)
m.begin()	Return an iterator pointing to the first pair.	O(1)
m.end()	Return an iterator pointing one past the last pair.	O(1)

Modifiers

m[key] = value;	Store value under key in map.	O(log n)
m.insert(pair)	Inserts the <key, value> pair into the map. Equivalent to the above operation.	O(log n)

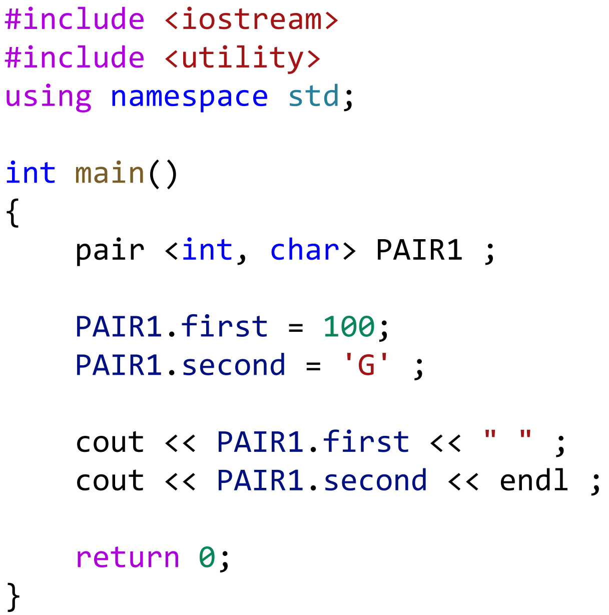
มารู้จัก Utility Library อยู่ในSTL ที่สำคัญอันนึงก่อน : Pair

Header

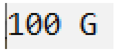
#include <utility>

Syntax

pair (data_type1, data_type2) Pair_name;



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

int main()
{
    pair <int, char> PAIR1 ;

    PAIR1.first = 100;
    PAIR1.second = 'G' ;

    cout << PAIR1.first << " " ;
    cout << PAIR1.second << endl ;

    return 0;
}
```



```
100 G
```

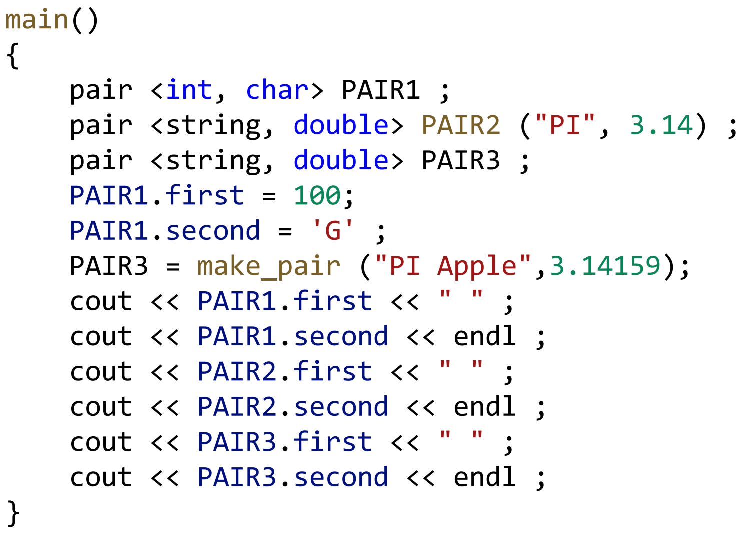
We can also initialize a pair.

pair (data_type1, data_type2) Pair_name (value1, value2) ;

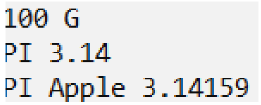
```
pair g1; //default
pair g2(1, 'a'); //initialized, different data type
pair g3(1, 10); //initialized, same data type
pair g4(g3); //copy of g3
```

Another way to initialize a pair is by using the make_pair() function.

```
g2 = make_pair(1, 'a');
```



```
main()
{
    pair <int, char> PAIR1 ;
    pair <string, double> PAIR2 ("PI", 3.14) ;
    pair <string, double> PAIR3 ;
    PAIR1.first = 100;
    PAIR1.second = 'G' ;
    PAIR3 = make_pair ("PI Apple",3.14159);
    cout << PAIR1.first << " " ;
    cout << PAIR1.second << endl ;
    cout << PAIR2.first << " " ;
    cout << PAIR2.second << endl ;
    cout << PAIR3.first << " " ;
    cout << PAIR3.second << endl ;
}
```

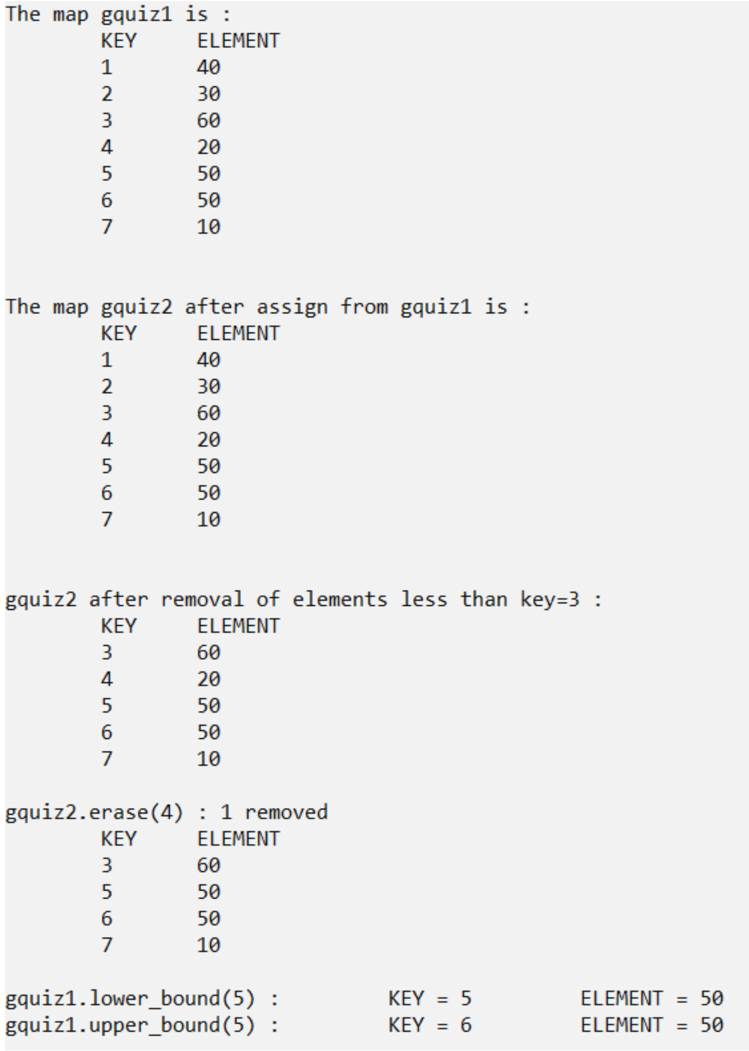


```
100 G
PI 3.14
PI Apple 3.14159
```

More: http://www.cplusplus.com/reference/utility/pair/

Containers : Map and Multimap

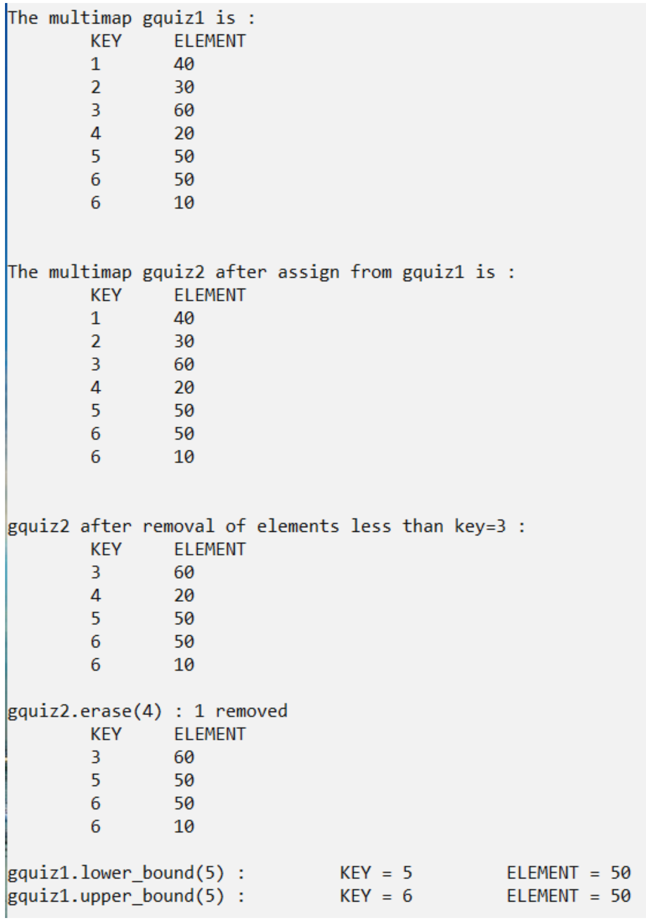
```cpp
#include <iostream>
#include <iterator>
#include <map>

using namespace std;

int main()
{

    // empty map container
    map<int, int> gquiz1;

    // insert elements in random order
    gquiz1.insert(pair<int, int>(1, 40));
    gquiz1.insert(pair<int, int>(2, 30));
    gquiz1.insert(pair<int, int>(3, 60));
    gquiz1.insert(pair<int, int>(4, 20));
    gquiz1.insert(pair<int, int>(5, 50));
    gquiz1.insert(pair<int, int>(6, 50));
    gquiz1.insert(pair<int, int>(7, 10));

    // printing map gquiz1
    map<int, int>::iterator itr;
    cout << "\nThe map gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr) {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }
    cout << endl;

    // assigning the elements from gquiz1 to gquiz2
    map<int, int> gquiz2(gquiz1.begin(), gquiz1.end());

    // print all elements of the map gquiz2
    cout << "\nThe map gquiz2 after"
         << " assign from gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }
    cout << endl;

    // remove all elements up to
    // element with key=3 in gquiz2
    cout << "\ngquiz2 after removal of"
            " elements less than key=3 : \n";
    cout << "\tKEY\tELEMENT\n";
    gquiz2.erase(gquiz2.begin(), gquiz2.find(3));
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }

    // remove all elements with key = 4
    int num;
    num = gquiz2.erase(4);
    cout << "\ngquiz2.erase(4) : ";
    cout << num << " removed \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }

    cout << endl;

    // lower bound and upper bound for map gquiz1 key = 5
    cout << "gquiz1.lower_bound(5) : "
         << "\tKEY = ";
    cout << gquiz1.lower_bound(5)->first << '\t';
    cout << "\tELEMENT = "
         << gquiz1.lower_bound(5)->second << endl;
    cout << "gquiz1.upper_bound(5) : "
         << "\tKEY = ";
    cout << gquiz1.upper_bound(5)->first << '\t';
    cout << "\tELEMENT = "
         << gquiz1.upper_bound(5)->second << endl;

    return 0;
}
```



```
The map gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        7       10


The map gquiz2 after assign from gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        7       10


gquiz2 after removal of elements less than key=3 :
        KEY     ELEMENT
        3       60
        4       20
        5       50
        6       50
        7       10

gquiz2.erase(4) : 1 removed
        KEY     ELEMENT
        3       60
        5       50
        6       50
        7       10

gquiz1.lower_bound(5) :         KEY = 5         ELEMENT = 50
gquiz1.upper_bound(5) :         KEY = 6         ELEMENT = 50
```

Containers : Map and Multimap

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

using namespace std;

int main()
{
    multimap <int, int> gquiz1;        // empty multimap container

    // insert elements in random order
    gquiz1.insert(pair <int, int> (1, 40));
    gquiz1.insert(pair <int, int> (2, 30));
    gquiz1.insert(pair <int, int> (3, 60));
    gquiz1.insert(pair <int, int> (4, 20));
    gquiz1.insert(pair <int, int> (5, 50));
    gquiz1.insert(pair <int, int> (6, 50));
    gquiz1.insert(pair <int, int> (6, 10));

    // printing multimap gquiz1
    multimap <int, int> :: iterator itr;
    cout << "\nThe multimap gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr)
    {
        cout  <<  '\t' << itr->first
              <<  '\t' << itr->second << '\n';
    }
    cout << endl;

    // assigning the elements from gquiz1 to gquiz2
    multimap <int, int> gquiz2(gquiz1.begin(),gquiz1.end());

    // print all elements of the multimap gquiz2
    cout << "\nThe multimap gquiz2 after assign from gquiz1 is : \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }
    cout << endl;

    // remove all elements up to element with value 30 in gquiz2
    cout << "\ngquiz2 after removal of elements less than key=3 : \n";
    cout << "\tKEY\tELEMENT\n";
    gquiz2.erase(gquiz2.begin(), gquiz2.find(3));
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }

    // remove all elements with key = 4
    int num;
    num = gquiz2.erase(4);
    cout << "\ngquiz2.erase(4) : ";
    cout << num << " removed \n" ;
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr)
    {
        cout << '\t' << itr->first
             << '\t' << itr->second << '\n';
    }

    cout << endl;

    //lower bound and upper bound for multimap gquiz1 key = 5
    cout << "gquiz1.lower_bound(5) : " << "\tKEY = ";
    cout << gquiz1.lower_bound(5)->first << '\t';
    cout << "\tELEMENT = " << gquiz1.lower_bound(5)->second << endl;
    cout << "gquiz1.upper_bound(5) : " << "\tKEY = ";
    cout << gquiz1.upper_bound(5)->first << '\t';
    cout << "\tELEMENT = " << gquiz1.upper_bound(5)->second << endl;

    return 0;

}
```



```
The multimap gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        6       10


The multimap gquiz2 after assign from gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        6       10


gquiz2 after removal of elements less than key=3 :
        KEY     ELEMENT
        3       60
        4       20
        5       50
        6       50
        6       10

gquiz2.erase(4) : 1 removed
        KEY     ELEMENT
        3       60
        5       50
        6       50
        6       10

gquiz1.lower_bound(5) :         KEY = 5         ELEMENT = 50
gquiz1.upper_bound(5) :         KEY = 6         ELEMENT = 50
```

Algorithm

Algorithm

Non-modifying sequence operations

1.std :: all_of : Test condition on all elements in range

2.std :: any_of : Test if any element in range fulfills condition

3.std :: none_of : Test if no elements fulfill condition

4.std :: for_each : Apply function to range

5.std :: find : Find value in range

6.std :: find_if : Find element in range

7.std :: find_if_not : Find element in range (negative condition)

8.std :: find_end : Find last subsequence in range

9.std :: find_first_of : Find element from set in range

10.std :: adjacent_find : Find equal adjacent elements in range

11.std :: count : Count appearances of value in range

12.std :: count_if : Return number of elements in range satisfying condition

13.std :: mismatch : Return first position where two ranges differ

14.std::equal : Test whether the elements in two ranges are equal

15.std :: is_permutation : Test whether range is permutation of another

16.std :: search : Search range for subsequence

17.std :: search_n : Search range for element

Modifying sequence operations

1.std :: copy : Copy range of elements

2.std :: copy_n : Copy elements

3.std :: copy_if : Copy certain elements of range

4.std :: copy_backward : Copy range of elements backward

5.std::move : Move range of elements

6.std :: move_backward : Move range of elements backward

7.std :: swap : Exchange values of two objects

8.std ::swap_ranges : Exchange values of two ranges

9.std :: iter_swap : Exchange values of objects pointed to by two iterators

10.std ::transform : Transform range

11.std ::replace : Replace value in range

12.std ::replace_if : Replace values in range

13.std :: replace_copy : Copy range replacing value

14.std :: replace_copy_if : Copy range replacing value

15.std ::fill : Fill range with value

16.std :: fill_n : Fill sequence with value

17.std ::generate : Generate values for range with function

18.std ::generate_n : Generate values for sequence with function

19.std ::remove : Remove value from range

20.std :: remove_if : Remove elements from range

21.remove_copy : Copy range removing value

22.remove_copy_if : Copy range removing values

23.std ::unique : Remove consecutive duplicates in range

24.std :: unique_copy : Copy range removing duplicates

25.std ::reverse : Reverse range

26.std :: reverse_copy : Copy range reversed

27.std :: rotate : Rotate left the elements in range

28.std :: rotate_copy : Copy range rotated left

29.std :: random_shuffle : Randomly rearrange elements in range

30.std :: shuffle : Randomly rearrange elements in range using generator

Partition Operations

1.std :: is_partitioned : Test whether range is partitioned

2.std :: partition : Partition range in two

3.std :: stable_partition : Partition range in two – stable ordering

4.partition_copy : Partition range into two

5.partition_point : Get partition point

Sorting

1.std :: sort : Sort elements in range

2.std :: stable_sort : Sort elements preserving order of equivalents

3.std :: partial_sort : Partially sort elements in range

4.std :: partial_sort_copy : Copy and partially sort range

5.std :: is_sorted : Check whether range is sorted

6.std :: is_sorted_until : Find first unsorted element in range

7.std :: nth_element : Sort element in range

Binary search (operating on partitioned/sorted ranges)

1.std :: lower_bound : Return iterator to lower bound

2.std :: upper_bound : Return iterator to upper bound

3.std :: equal_range : Get subrange of equal elements

4.std :: binary_search : Test if value exists in sorted sequence

Merge (operating on sorted ranges)

1.std :: merge : Merge sorted ranges

2.std :: inplace_merge : Merge consecutive sorted ranges

3.std :: includes : Test whether sorted range includes another sorted range

4.std :: set_union : Union of two sorted ranges

5.std :: set_intersection : Intersection of two sorted ranges

6.std :: set_difference : Difference of two sorted ranges

7.std :: set_symmetric_difference : Symmetric difference of two sorted ranges

Heap Operations

1.std :: push_heap : Push element into heap range

2.std :: pop_heap : Pop element from heap range

3.std :: make_heap : Make heap from range

4.std :: sort_heap : Sort elements of heap

5.std :: is_heap : Test if range is heap

6.std :: is_heap_until : Find first element not in heap order

7.std :: max : Return the largest

8.std :: minmax : Return smallest and largest elements

9.std :: min_element : Return smallest element in range

10.std :: max_element : Return largest element in range

11.std :: minmax_element : Return smallest and largest elements in range

Other Operations

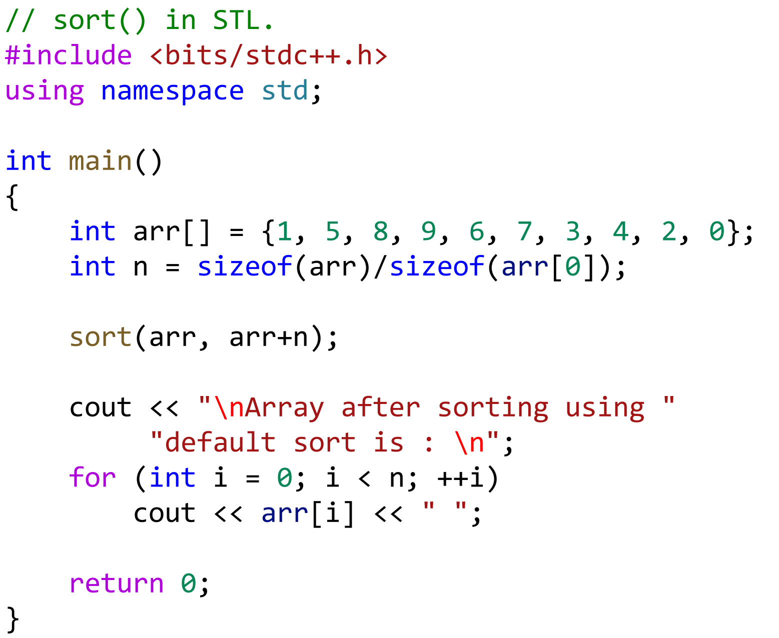
1.std :: lexicographical_compare : Lexicographical less-than comparison

2.std :: next_permutation : Transform range to next permutation

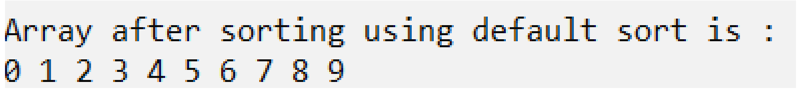
3.std :: prev_permutation : Transform range to previous permutation

Algorithm : Important

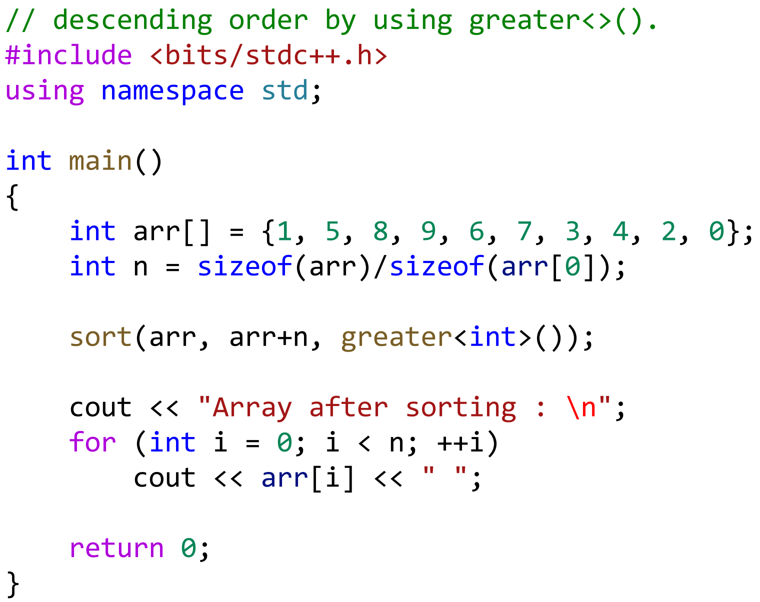
1. sort(first_iterator, last_iterator) – To sort the given vector.



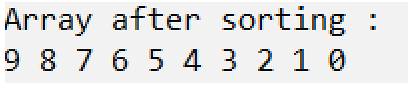
```cpp
// sort() in STL.
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    sort(arr, arr+n);

    cout << "\nArray after sorting using "
         "default sort is : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```



```
Array after sorting using default sort is :
0 1 2 3 4 5 6 7 8 9
```



```cpp
// descending order by using greater<>().
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);

    sort(arr, arr+n, greater<int>());

    cout << "Array after sorting : \n";
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";

    return 0;
}
```

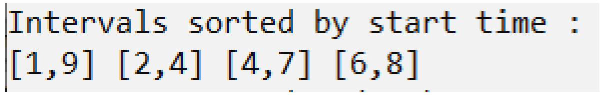


```
Array after sorting :
9 8 7 6 5 4 3 2 1 0
```

Algorithm : Important

1. **sort(first_iterator, last_iterator)** – To sort the given vector.

```cpp
// our own comparator
#include<bits/stdc++.h>
using namespace std;

// An interval has a start time and end time
struct Interval
{
    int start, end;
};

// Compares two intervals according to staring times.
bool compareInterval(Interval i1, Interval i2)
{
    return (i1.start < i2.start);
}

int main()
{
    Interval arr[] =  { {6,8}, {1,9}, {2,4}, {4,7} };
    int n = sizeof(arr)/sizeof(arr[0]);

    // sort the intervals in increasing order of
    // start time
    sort(arr, arr+n, compareInterval);

    cout << "Intervals sorted by start time : \n";
    for (int i=0; i<n; i++)
       cout << "[" << arr[i].start << "," << arr[i].end
            << "] ";

    return 0;
}
```



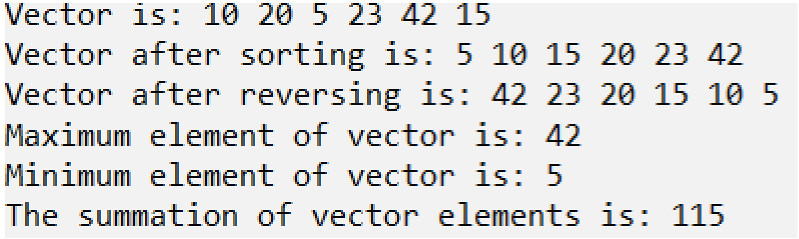
```
Intervals sorted by start time :
[1,9] [2,4] [4,7] [6,8]
```

Algorithm : Important

2.reverse(first_iterator, last_iterator) – To reverse a vector.
3.*max_element (first_iterator, last_iterator) – To find the maximum element of a vector.
4.*min_element (first_iterator, last_iterator) – To find the minimum element of a vector.
5.accumulate(first_iterator, last_iterator, initial value of sum) – Does the summation of vector elements

```cpp
// A C++ program to demonstrate working of sort(),
// reverse()
#include <algorithm>
#include <iostream>
#include <vector>
#include <numeric> //For accumulate operation
using namespace std;
int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23 ,42 , 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is: ";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";
    // Sorting the Vector in Ascending order
    sort(vect.begin(), vect.end());
    cout << "\nVector after sorting is: ";
    for (int i=0; i<n; i++)
       cout << vect[i] << " ";
    // Reversing the Vector
    reverse(vect.begin(), vect.end());
```

```cpp
    cout << "\nVector after reversing is: ";
    for (int i=0; i<6; i++)
        cout << vect[i] << " ";
    cout << "\nMaximum element of vector is: ";
    cout << *max_element(vect.begin(), vect.end());
    cout << "\nMinimum element of vector is: ";
    cout << *min_element(vect.begin(), vect.end());
    // Starting the summation from 0
    cout << "\nThe summation of vector elements is: ";
    cout << accumulate(vect.begin(), vect.end(), 0);

    return 0;
}
```

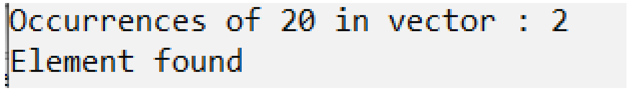


```
Vector is: 10 20 5 23 42 15
Vector after sorting is: 5 10 15 20 23 42
Vector after reversing is: 42 23 20 15 10 5
Maximum element of vector is: 42
Minimum element of vector is: 5
The summation of vector elements is: 115
```

Algorithm : Important

6. count(first_iterator, last_iterator,x) – To count the occurrences of x in vector.
7. find(first_iterator, last_iterator, x) – Points to last address of vector ((name_of_vector).end()) if element is not present in vector

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

int main()
{
    // Initializing vector with array values
    int arr[] = {10, 20, 5, 23 ,42, 20, 15};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Occurrences of 20 in vector : ";

    // Counts the occurrences of 20 from 1st to
    // last element
    cout << count(vect.begin(), vect.end(), 20);

    // find() returns iterator to last address if
    // element not present
    find(vect.begin(), vect.end(),5) != vect.end()?
                         cout << "\nElement found":
                     cout << "\nElement not found";

    return 0;
}
```

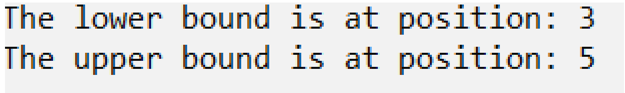


```
Occurrences of 20 in vector : 2
Element found
```

Algorithm : Important

8. binary_search(first_iterator, last_iterator, x) – Tests whether x exists in sorted vector or not.
9. lower_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value not less than ‘x’.
10. upper_bound(first_iterator, last_iterator, x) – returns an iterator pointing to the first element in the range [first,last) which has a value greater than ‘x’.

```cpp
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Sort the array to make sure that lower_bound()
    // and upper_bound() work.
    sort(vect.begin(), vect.end());

    // Returns the first occurrence of 20
    auto q = lower_bound(vect.begin(), vect.end(), 20);

    // Returns the last occurrence of 20
    auto p = upper_bound(vect.begin(), vect.end(), 20);

    cout << "The lower bound is at position: ";
    cout << q-vect.begin() << endl;

    cout << "The upper bound is at position: ";
    cout << p-vect.begin() << endl;

    return 0;
}
```

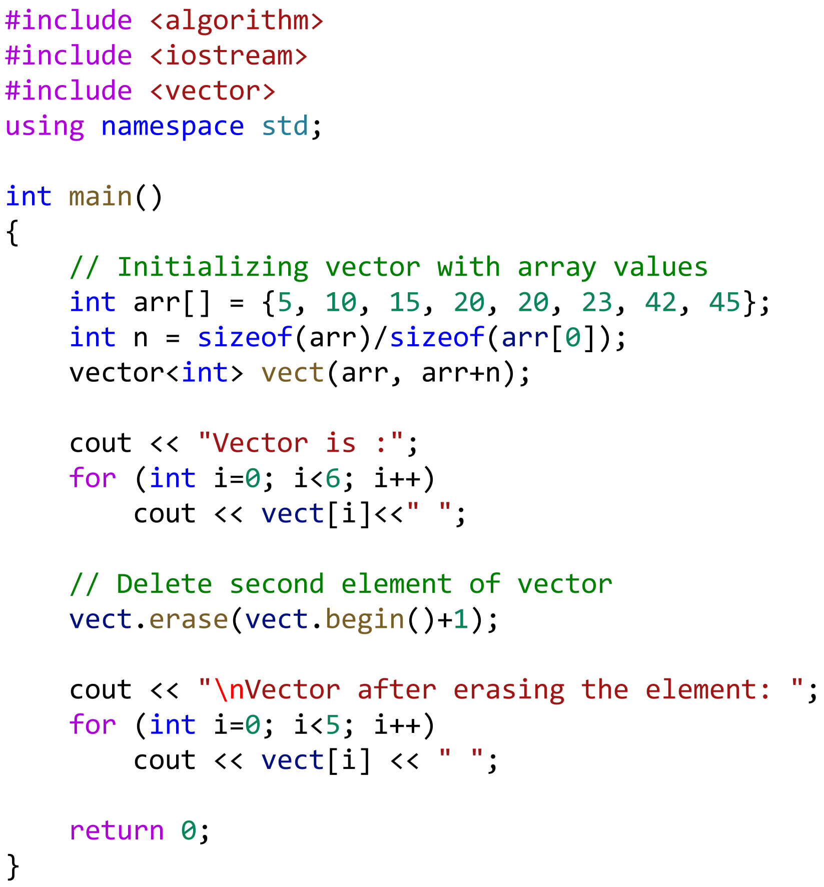


```
The lower bound is at position: 3
The upper bound is at position: 5
```

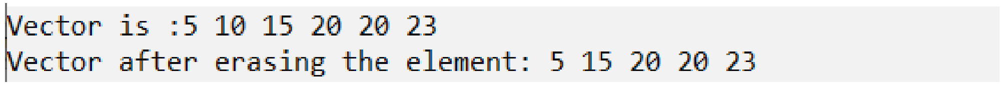
Algorithm : Important

11. arr.erase(position to be deleted) – This erases selected element in vector and shifts and resizes the vector elements accordingly.

12. arr.erase(arr.begin(), arr.begin()+n) – This erases the first n element



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

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Vector is :";
    for (int i=0; i<6; i++)
        cout << vect[i]<<" ";

    // Delete second element of vector
    vect.erase(vect.begin()+1);

    cout << "\nVector after erasing the element: ";
    for (int i=0; i<5; i++)
        cout << vect[i] << " ";

    return 0;
}
```

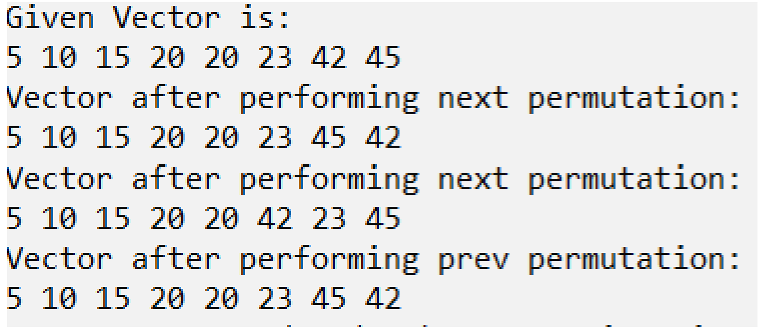


```
Vector is :5 10 15 20 20 23
Vector after erasing the element: 5 15 20 20 23
```

Algorithm : Important

13. next_permutation(first_iterator, last_iterator) – This modified the vector to its next permutation.
14. prev_permutation(first_iterator, last_iterator) – This modified the vector to its previous permutation.

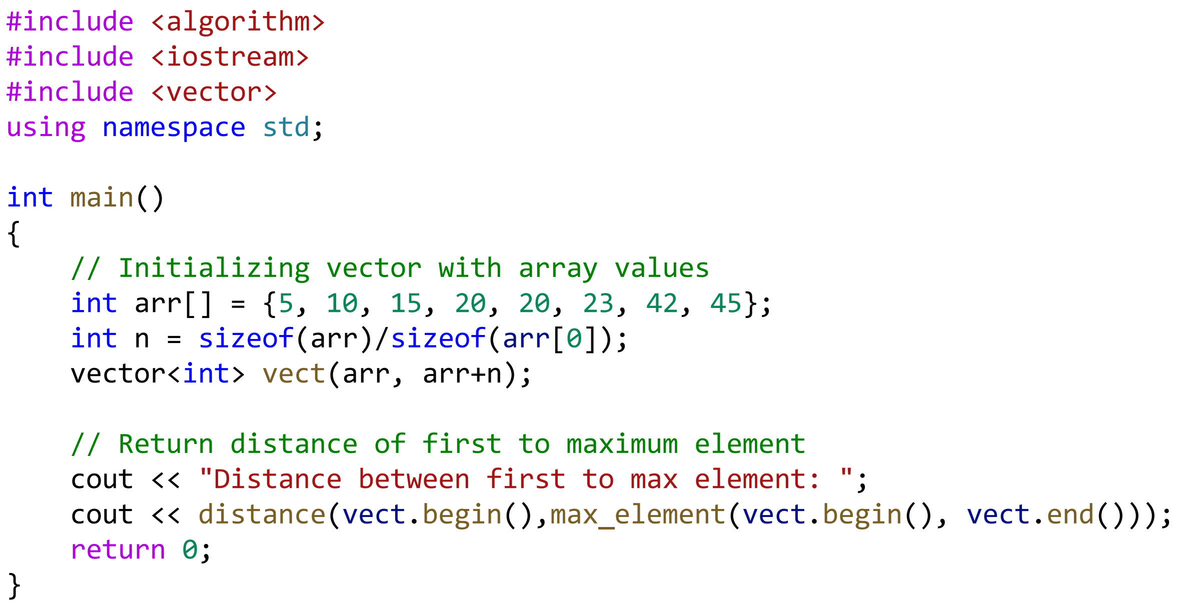
```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    cout << "Given Vector is:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    // modifies vector to its next permutation order
    next_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing next permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    next_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing next permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    prev_permutation(vect.begin(), vect.end());
    cout << "\nVector after performing prev permutation:\n";
    for (int i=0; i<n; i++)
        cout << vect[i] << " ";

    return 0;
}
```



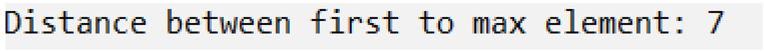
```
Given Vector is:
5 10 15 20 20 23 42 45
Vector after performing next permutation:
5 10 15 20 20 23 45 42
Vector after performing next permutation:
5 10 15 20 42 23 45
Vector after performing prev permutation:
5 10 15 20 20 23 45 42
```

Algorithm : Important

15. distance(first_iterator,desired_position) – It returns the distance of desired position from the first iterator.This function is very useful while finding the index.



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

int main()
{
    // Initializing vector with array values
    int arr[] = {5, 10, 15, 20, 20, 23, 42, 45};
    int n = sizeof(arr)/sizeof(arr[0]);
    vector<int> vect(arr, arr+n);

    // Return distance of first to maximum element
    cout << "Distance between first to max element: ";
    cout << distance(vect.begin(),max_element(vect.begin(), vect.end()));
    return 0;
}
```



```
Distance between first to max element: 7
```