**Priority Queues**

Class priority\_queue<> implements a queue from which elements are read according to their priority.

header file

#include <queue>

The class priority\_queue is defined as follows:

namespace std {

template <

typename T,

typename Container = vector<T>,

typename Compare = less<typename Container::value\_type>>

class priority\_queue;

}

**T:** type of the elements

**Container:** Optional template parameter defines the container that the priority queue uses internally for its elements. Default container is a vector.

**Compare:** Defines the sorting criterion used to find the next element with the highest priority. Default is <.

The priority queue implementation simply maps the operations into appropriate calls of the container that is used internally. You can use any sequence container class that provides random-access iterators and the member functions front(), push\_back(), and pop\_back(). Random access is necessary for sorting the elements, which is performed by the heap algorithms of the STL.

std::priority\_queue<float, std::deque<float>> pbuffer;

To define your own sorting criterion, you must pass a function, a function object, or a lambda as a binary predicate that is used by the sorting algorithms to compare two elements.

std::priority\_queue<float, std::vector<float>, std::greater<float>> pbuffer;

# The Core Interface

**push()** inserts an element into the priority queue.

**top()** returns the next element in the priority queue without removing it.

**pop()** removes an element from the priority queue but does not return it.

Call both functions top() and pop() to process and remove the next element from the priority queue.

However, the next element is not the first inserted element. Rather, it is the element that has the highest priority.

Member functions size() and empty() are provided to check whether the queue contains elements.

# Class priority\_queue<> in Detail

The priority queue uses the STL’s heap algorithms:

namespace std {

template <

typename T,

typename Container = vector<T>,

typename Compare = less<typename Container::value\_type> >

class priority\_queue {

protected:

Compare comp; // sorting criterion

Container c; // container

public:

// constructors

explicit priority\_queue(const Compare& cmp = Compare(), const Container& cont = Container()) : comp(cmp), c(cont) {

make\_heap(c.begin(),c.end(),comp);

}

void push(const value\_type& x) {

c.push\_back(x);

push\_heap(c.begin(),c.end(),comp);

}

void pop() {

pop\_heap(c.begin(),c.end(),comp);

c.pop\_back();

}

bool empty() const { return c.empty(); }

size\_type size() const { return c.size(); }

const value\_type& top() const { return c.front(); }

...

};

}

Note that, unlike other container adapters, no comparison operators are defined.

# Example

#include <iostream>

#include <queue>

using namespace std;

int main() {

priority\_queue<float> q;

// insert three elements into the priority queue

q.push(66.6);

q.push(22.2);

q.push(44.4);

// read and print two elements

cout << q.top() << " "; q.pop();

cout << q.top() << endl; q.pop();

// insert three more elements

q.push(11.1);

q.push(55.5);

q.push(33.3);

// skip one element

q.pop();

// pop and print remaining elements

while (!q.empty()) {

cout << q.top() << " "; q.pop();

}

cout << endl;

return 0;

}

Output:

66.6 44.4

33.3 22.2 11.1

# END