A decent implementation of a priority queue that uses the standard library can be found in the standard library itself.

The standard defines a class template called “priority_queue” (see §23.6.4). This class template represents a canonical implementation of a priority queue - a binary heap built on the top on an array.

Here’s my simplified version of it (I've omitted type definitions and constructors, but everything else is pretty much a direct quote from the standard):

```
template <class T, class Container = std::vector<T>,
          class Compare = std::less<T> >
class priority_queue {
protected:
    Container c;
    Compare comp;

public:
    explicit priority_queue(const Container& c_  = Container(),
                            const Compare& comp_ = Compare())
        : c(c_), comp(comp_)
    {
        std::make_heap(c.begin(), c.end(), comp);
    }

    bool empty()       const { return c.empty(); }
    std::size_t size() const { return c.size(); }

    const T& top()     const { return c.front(); }

    void push(const T& x)
    {
        c.push_back(x);
        std::push_heap(c.begin(), c.end(), comp);
    }

    void pop()
    {
        std::pop_heap(c.begin(), c.end(), comp);
        c.pop_back();
    }
};
```

This code is pretty straightforward, especially if you're already familiar with binary heaps.

Template parameter `T` specifies a type of elements stored in the queue. The `Container`parameter specifies the container that is used to store elements (it's `std::vector` by default, but any sequence container with random access iterator and `front()`, `push_back()` and `pop_back()` operations can be used).
Lastly, `Compare` specifies the comparator - `std::less` gives you a max-oriented priority queue, `std::greater` gives you a min-oriented one.

In constructor `std::make_heap()` converts the contents of a given container into a heap. The way it does this is implementation-defined, but generally a variation of linear-time sift-down-based algorithm is used. If you're not familiar with binary heap algorithms, consider consulting with your favorite book. Or take a look at the actual implementation (search for the `make_heap` method).

The `top()` method returns you a top element.

The `push()` method adds an element to the queue. First, `push()` appends given element to the end the

container. Then `std::push_heap()` propagates it up the queue until it's in the right place using the sift-up algorithm. Again, you might want to consult with a book orthe implementation (look for `push_heap`).

Finally, the `pop()` method removes the top element from the queue. The call to `std::pop_heap()` moves the top element to the end of the container by swapping it with the last one and then "heapifies" the rest of the contents using the good old sift-down. After that `pop_back()` removes the last element.

And this is pretty much how a priority queue is implemented in C++.