C++ priority queues

A priority queue is an abstract data type that captures the idea of a container whose elements have "priorities" attached to them. An element of highest priority always appears at the front of the queue. If that element is removed, the next highest priority element advances to the front.

The C++ standard library defines a class template **priority_queue**, with the following operations:

- push: Insert an element into the prioity queue.
- top: Return (without removing it) a highest priority element from the priority queue.
- pop: Remove a highest priority element from the priority queue.
- size: Return the number of elements in the priority queue.
- empty: Return true or false according to whether the priority queue is empty or not.

The following code snippet shows how to construct two priority queues, one that can contain integers and another one that can contain character strings:

```
#include <queue>

priority_queue<int> q1;
priority_queue<string> q2;
```

The following is an example of priority queue usage:

```
#include <string>
#include <queue>
#include <iostream>
```

```
using namespace std;  // This is to make available the
names of things defined in the standard library.

int main()
{
    piority_queue<string> pq; // Creates a priority
queue pq to store strings, and initializes the queue
to be empty.

    pq.push("the quick");
    pq.push("fox");
    pq.push("jumped over");
    pq.push("the lazy dog");

    // The strings are ordered inside the priority
queue in lexicographic (dictionary) order:
    // "fox", "jumped over", "the lazy dog", "the
quick"
    //  The lowest priority string is "fox", and the
highest priority string is "the quick"

    while (!pq.empty()) {
       cout << pq.front() << endl;  // Print highest
priority string
       pq.pop();                    // Remmove highest
priority string
    }

    return 0;
}
```

The output of this program is:

```
the quick
```

```
the lazy dog
jumped over
fox
```

Since a queue follows a priority discipline, the strings are printed from highest to lowest priority.

Sometimes one needs to create a priority queue to contain user defined objects. In this case, the priority queue needs to know the comparison criterion used to determine which objects have the highest priority. This is done by means of a *function object* beloging to a class that overloads the operator (). The overloaded () acts as < for the purpose of determining priorities. For example, suppose we want to create a priority queue to store Time objects. A Time object has three fields: hours, minutes, seconds:

```
struct Time {
    int h;
    int m;
    int s;
};

class CompareTime {
    public:
    bool operator()(Time& t1, Time& t2) // Returns
true if t1 is earlier than t2
    {
       if (t1.h < t2.h) return true;
       if (t1.h == t2.h && t1.m < t2.m) return true;
       if (t1.h == t2.h && t1.m == t2.m && t1.s <
t2.s) return true;
       return false;
    }
}
```

A priority queue to store times acoording the the above comparison criterion would be defined as follows:

```
priority_queue<Time, vector<Time>, CompareTime> pq;
```

Here is a complete program:

```
#include <iostream>
#include <queue>
#include <iomanip>

using namespace std;

struct Time {
    int h; // >= 0
    int m; // 0-59
    int s; // 0-59
};

class CompareTime {
public:
    bool operator()(Time& t1, Time& t2)
    {
       if (t1.h < t2.h) return true;
       if (t1.h == t2.h && t1.m < t2.m) return true;
       if (t1.h == t2.h && t1.m == t2.m && t1.s <
t2.s) return true;
       return false;
    }
};

int main()
{
    priority_queue<Time, vector<Time>, CompareTime>
pq;
```

```
    // Array of 4 time objects:

    Time t[4] = { {3, 2, 40}, {3, 2, 26}, {5, 16, 13},
{5, 14, 20}};

    for (int i = 0; i < 4; ++i)
       pq.push(t[i]);

    while (! pq.empty()) {
       Time t2 = pq.top();
       cout << setw(3) << t2.h << " " << setw(3) <<
t2.m << " " <<
       setw(3) << t2.s << endl;
       pq.pop();
    }

    return 0;
}
```

The program prints the times from latest to earliest:

```
    5  16  13
    5  14  20
    3   2  40
    3   2  26
```

If we wanted earliest times to have the highest priority, we would redefine `CompareTime` like this:

```
class CompareTime {
public:
    bool operator()(Time& t1, Time& t2) // t2 has
highest prio than t1 if t2 is earlier than t1
    {
       if (t2.h < t1.h) return true;
       if (t2.h == t1.h && t2.m < t1.m) return true;
       if (t2.h == t1.h && t2.m == t1.m && t2.s <
```

```
t1.s) return true;
        return false;
    }
};
```