How to use the priority queue STL for objects?

```
class Person
{
public:
    int age;
};
```

I want to store objects of the class Person in a priority queue.

```
priority_queue< Person, vector<Person>, ??? >
```

I think I need to define a class for the comparison thing, but I am not sure about it.

Also, when we write,

```
priority_queue< int, vector<int>, greater<int> >
```

How does the greater work?

c++ stl

asked Oct 23 '13 at 7:38

user2441151
363 1 3 13

Similar post here – Rick Smith Mar 13 at 23:34

3 Answers

You need to provide a valid strict weak ordering comparison for the type stored in the queue, `Person` in this case. The default is to use `std::less<T>`, which resolves to something equivalent to `operator<`. This relies on it's own stored type having one. So if you were to implement

```
bool operator<(const Person& lhs, const Person& rhs);
```

it should work without any further changes. The implementation could be

```
bool operator<(const Person& lhs, const Person& rhs)
{
  return lhs.age < rhs.age;
}
```

If the the type does not have a natural "less than" comparison, it would make more sense to provide your own predicate, instead of the default `std::less<Person>`. For example,

```
struct LessThanByAge
{
  bool operator()(const Person& lhs, const Person& rhs) const
  {
    return lhs.age < rhs.age;
  }
};
```

then instantiate the queue like this:

```
std::priority_queue<Person, std::vector<Person>, LessThanByAge> pq;
```

Concerning the use of `std::greater<Person>` as comparator, this would use the equivalent of `operator>` and have the effect of creating a queue with the priority inverted WRT the default case. It would require the presence of an `operator>` that can operate on two `Person` instances.

edited Apr 21 '14 at 10:22

answered Oct 23 '13 at 7:41

juanchopanza
148k 13 169 288

4 While this answer is correct, I dislike the use of `operator<` here. `operator<` implements the default comparison for a type, which, in my experience, is rarely what you want. I think the approach Mike describes in his answer is almost always preferable. – Björn Pollex Oct 23 '13 at 7:48

And there you go and edit it :) - +1! – Björn Pollex Oct 23 '13 at 7:48

1 @BjörnPollex Agreed. I was adding something about that. In a class with only one data member, the operator *might* have made sense. – juanchopanza Oct 23 '13 at 7:48

Worthy to note: implementing `bool YourClass::operator <(const YourClass&) const` will also allow transparent use of the default comparator, `std::less<T>` . Not as flexible, but functional when that is all you need. (and +1). – WhozCraig Oct 23 '13 at 7:53

Thanks for the answer. I can overload '<' operator even if the class has multiple members, right? – user2441151 Oct 23 '13 at 8:14

You would write a comparator class, for example:

```
struct CompareAge {
    bool operator()(Person const & p1, Person const & p2) {
        // return "true" if "p1" is ordered before "p2", for example:
        return p1.age < p2.age;
    }
};
```

and use that as the comparator argument:

```
priority_queue<Person, vector<Person>, CompareAge>
```

Using `greater` gives the opposite ordering to the default `less` , meaning that the queue will give you the lowest value rather than the highest.

answered Oct 23 '13 at 7:43

Mike Seymour
177k 10 220 408

Thanks for the answer. :) – user2441151 Oct 23 '13 at 8:12

It's possible to pass "comparator objects" instead of comparator classes? (in order to parametrize it and obtain more flexibility) – castarco Nov 23 '14 at 19:40

@castarco yes, you can pass a specific comparator object as a constructor argument. – Mike Seymour Nov 23 '14 at 19:53

This piece of code may help..

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

class node{
public:
    int age;
    string name;
    node(int a, string b){
        age = a;
        name = b;
    }
};

bool operator<(const node& a, const node& b) {

    node temp1=a,temp2=b;
    if(a.age != b.age)
        return a.age > b.age;
    else{
        return temp1.name.append(temp2.name) > temp2.name.append(temp1.name);
    }
}

int main(){
    priority_queue<node> pq;
    node b(23,"prashantandsoon..");
    node a(22,"prashant");
    node c(22,"prashantonly");
    pq.push(b);
    pq.push(a);
    pq.push(c);
```

```
    int size = pq.size();
    for (int i = 0; i < size; ++i)
    {
        cout<<pq.top().age<<" "<<pq.top().name<<"\n";
        pq.pop();
    }
}
```

Output:

```
22 prashantonly
22 prashant
23 prashantandsoon..
```

answered Oct 2 at 8:36

Navneet Swaminathan

11 3