ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Priority Queues in the STL

Goodrich chapter 7

ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Priority Queues - Introduction

- A Priority Queue is a queue with some extra features
  - Each data item on the queue is assigned a priority.
  - If all the items have the same priority, the queue would work like any other queue.
  - But when the priorities differ, the item with the highest priority is always the next one removed.
  - If more than one item has the same priority, the order in which they are removed is undefined.

- Commonly a priority queue is implemented where the value of the item determines its priority.
  - So the highest value is always at the head of the queue
  - it will be the next one removed.

- Highest *value* doesn't have to mean highest *priority*: e.g. the highest priority may be to look after employee on lowest salary. (if only ...)
  - we maintain the list in sorted order, with the lowest salary at the head of the queue, then next one removed from the queue would always have highest priority.
  - We would have to do some work when inserting an Employee to search along the list in order to identify the position at which to make the insertion
  - But we could then be sure that the one at the top was the highest priority.

ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Priority queue in the STL

- **`The template class priority_queue`** is defined in the `<queue>` header file. It is another adapter class, built on top of the sequential containers
- a Priority queue is (by default) implemented internally by using a vector, and using the standard < operator to determine priority we can declare a priority queue of `ints` which uses these defaults as **`priority_queue<int>`**
- The default container used in the implementation (vector), can as usual be overridden
  - Any sequential container that uses random access iterators and member functions `front()` `push_back()` and `pop_back()` can be used for the underlying container
  - **`priority_queue<int, deque<int> >`**
    will provide a priority queue of `ints` where the underlying implementation uses a `deque` (of ints)

Comparator for the priority queue

- As usual we can supply some other function to use to determine 'less than' (which in a priority queue of course means "lower priority"
- But it gets a little bit complicated, as we need to wrap it in a 'Comparator class'

- `priority_queue<Employee, deque<Employee> , EmpComp>`

The comparator class: will need to be declared a friend by Employee class

```
class EmpComp {
public:
  bool operator()(const Employee& emp1, const Employee& emp2)
    {
      return emp1.salary > emp2.salary;  //remember, we have
      chosen that low priority means high salary!!
    }
};
```

Example code which uses a priority queue

```
class Star {

public:
    double intensity() const;
private:
    string identifier;
    double distance;
};

class StarCompare {
public:
    bool operator()( const Star& star1,   const Star& star2);
};

bool StarCompare::operator()( const Star& star1,   const Star& star2)
{
    return (  star1.intensity() <  star2.intensity() );
}
…
//declaring a priority queue of stars in our code
priority_queue<Star, vector<Star>, StarCompare> galaxy;
```

There would be a problem if we didn't declare this is as a const member function. Why??

Alternative to a comparator class

- We could overload the regular 'less than' operator in the Star class, then it would be used as the default behaviour for <
  - then we could declare a pq of Stars as `priority_queue<Star>`

```
class Star
{
public:
   Star(string id, double dist);
   string getIdentifier();
   double intensity() const;
   bool operator< (const Star& star2) const;
private:
   string identifier;
   double distance;
};

bool Star::operator<(const Star& star2) const
{
   return ( intensity() < star2.intensity() );
}
```

ITT DUBLIN
Institute of Technology Tallaght
Institiúid Teicneolaíochta Tamhlacht

Using an STL priority queue

Public Member functions

- Constructor
  - Construct priority queue
- `bool empty() const;`
  - Test whether container is empty
- `int size() const;`
  - Return size
- `const value_type& top ( ) const;`
  - Access top element
- `void push ( const T& x );`
  - Insert element
- `void pop()`
  - Remove top element