Gigi's Computer Corner

Daniel D'Agostino's Information Technology website

Main
Home
News
Site Map
About Me
Contact

IT Development
Computer Logic
Data Structures
Databases
Web Development
General Programming
Network Programming
Game Development

Specialised Topics
Natural Language P.
Physically Based R.
Email Technology

IT Administration
Hardware
Security

Other Info
Attitude
Maths & Physics
Operating Systems
Game Info
Miscellaneous

How to use the C++ STL Priority Queue

By Daniel D'Agostino, 19th January 2013

Overview

A **priority queue** is a queue data structure that has the particular property of being sorted by priority. You can decide what priority to give items depending on the data type - more on this in a minute.

The C++ Standard Template Library (STL) includes a convenient `std::priority_queue` class template in the `queue` header file.

A simple priority queue of integers

The following code sample illustrates how to implement a priority queue of integers. The integer value is used by default as a priority. The queue is sorted automatically as new entries are added.

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

int main(int argc, char ** argv)
{
	std::priority_queue<int> queue;

	queue.push(100);
	queue.push(300);
	queue.push(50);
	queue.push(150);

	while (!queue.empty())
	{
		std::cout << queue.top() << std::endl;
		queue.pop();
	}

	system("pause");

	return 0;
}
```

The output of the above program is as follows:

```
300
150
100
50
Press any key to continue . . .
```

A priority queue with a custom class

In practical situations, it is often not very useful to simply maintain a priority queue of just integers. We often have some particular class, and we want to give each instance a priority (computed based on some internal state).

Let's say we have a class called Toast, composed of a certain amount of bread and butter:

```
class Toast
{
public:
	int bread;
	int butter;
```

Recent Projects
Minesweeper Legacy
Magnetic Pool
IMAPTalk
Lilly Notes

Old Projects
Ultima 1 Revenge
Deposits System
Picaxo Image Viewer
Null Neuron Interpreter

Related Sites
Gigi Labs
Programmer's Ranch
Gigi on IT
BSc IT page
Gigi on Games
Dino's Ultima Page

ICT in Malta
Maltese ICT Sites
Tech Events

Friends
Tech Spark
WiredMash

Online Tools
Your IP Address
Base Converter
Regex Tester

```
        Toast(int bread, int butter)
                : bread(bread), butter(butter)
        {

        }
};
```

It is easy to sort integers, but how do you sort Toast? We need to offer C++ a way to compare one Toast instance to another. This is done by creating a structure implementing an `operator()` and effectively doing a *less-than* comparison. A StackOverflow question and answer shows how it's done, and the code needed for sorting our Toast is below.

```
struct ToastCompare
{
        bool operator()(const Toast &t1, const Toast &t2) const
        {
                int t1value = t1.bread * 1000 + t1.butter;
                int t2value = t2.bread * 1000 + t2.butter;
                return t1value < t2value;
        }
};
```

We can now pass the `TestCompare` class to the `priority_queue` to tell it how to sort the Toast. Sample code is below.

```
#include <iostream>
#include <queue>
#include <vector>

#include "Toast.h"

using std::priority_queue;
using std::vector;
using std::cout;
using std::endl;

int main(int argc, char ** argv)
{
        Toast toast1(2, 200);
        Toast toast2(1, 30);
        Toast toast3(1, 10);
        Toast toast4(3, 1);

        //priority_queue<Toast> queue;
        priority_queue<Toast, vector<Toast>, ToastCompare> queue;

        queue.push(toast1);
        queue.push(toast2);
        queue.push(toast3);
        queue.push(toast4);

        while (!queue.empty())
        {
                Toast t = queue.top();
                cout << "bread " << t.bread << " butter " << t.butter << std::endl;
                queue.pop();
        }

        system("pause");

        return 0;
}
```

If we used the simple `priority_queue` declaration (the line that is commented out), we would end up with a bunch of errors because of C++ not knowing how to compare the `Toast` instances.

Instead, we pass three template arguments: the `Toast` itself, a vector of `Toast`, and the `ToastCompare` class to tell C++ how to compare `Toast` instances. The second template argument (the vector) is there because the C++ STL priority queue is actually a container adapter - it uses an underlying data structure to store elements, and the default is a `vector`.

The output for the above program is given below:

```
bread 3 butter 1
bread 2 butter 200
bread 1 butter 30
bread 1 butter 10
Press any key to continue . . .
```

Further reading

- std::priority_queue - good reference for background and operations
- C++ priority queues - some good background and usage examples