# Template Queue class

Write a template Queue class that uses a base class that contains the code and data that does not depend on the type of elements to be stored in the Queue. In other words, the base class is ***not*** a template class, but the derived class Queue ***is*** a template class.

Your queue should take an initial capacity parameter that determines the capacity of your queue storage, i.e., how many elements you can store in the queue.

You **cannot use any STL container** class as the internal storage in the Queue. Instead, you should have your own array based storage, which means that your storage data member would look something like

```
T  *  m_Array;
```

where T is the template parameter passed in to your Queue class. For example: `Queue< int >` would mean that `T` represents an `int` in your class.

O**ptional feature**: You can make the Queue dynamically resizable, in which case the initial capacity parameter tells you how big the queue is to start with, and if you add more elements, then the queue should grow to accommodate the additional elements.

The member methods that you should add are:

Constructor
Push:  takes an element to put onto the queue.
Pop: returns the top element from the queue
IsEmpty: return true if empty, false otherwise.
Size: returns the number of elements currently in the queue.
Assignment operator that takes another queue of the same type as a parameter.
Assignment operator that takes another queue of a **different** template type as a parameter.

You should also add any other methods that you think are necessary.

# Exceptions

Define a simple exception hierarchy of your choice (try to keep the inheritance hierarchy less than or equal to 3 levels deep). Your methods push and pop should throw these exceptions… for example, the Pop method would throw a QueueEmpty exception if the queue is empty. The Push method would throw a QueueFull exception if it was full… now, if you decide to implement the optional feature mentioned above, then you would dynamically resize the queue if it became full, which means you would not throw a QueueFull kind of exception.

These exceptions should be caught in your class driver (where you are using the Queue class).

Your base exception class should define a pure virtual method called what() that returns a const char * which is a very brief description of the exception. The derived exception classes then should override this method.

When you catch the exception, print the string returned from what() to standard out (or standard error).