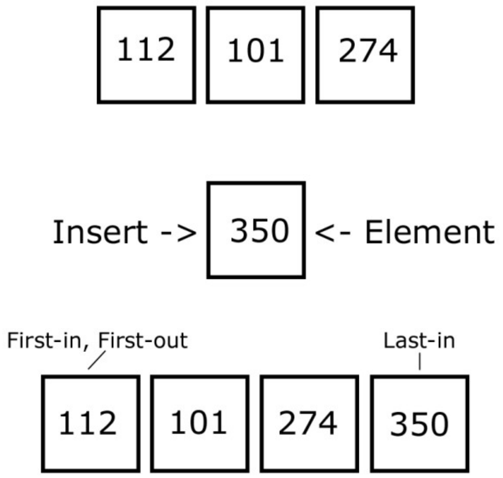
GAME2001 Data Structures and Algorithms

Fall 2020

Week 9
Stacks and Queues II

Queues

- has a first-in, first-out order
  - Inserted at the front and removed from the back
- has restricted access
- only one element is accessible at a time



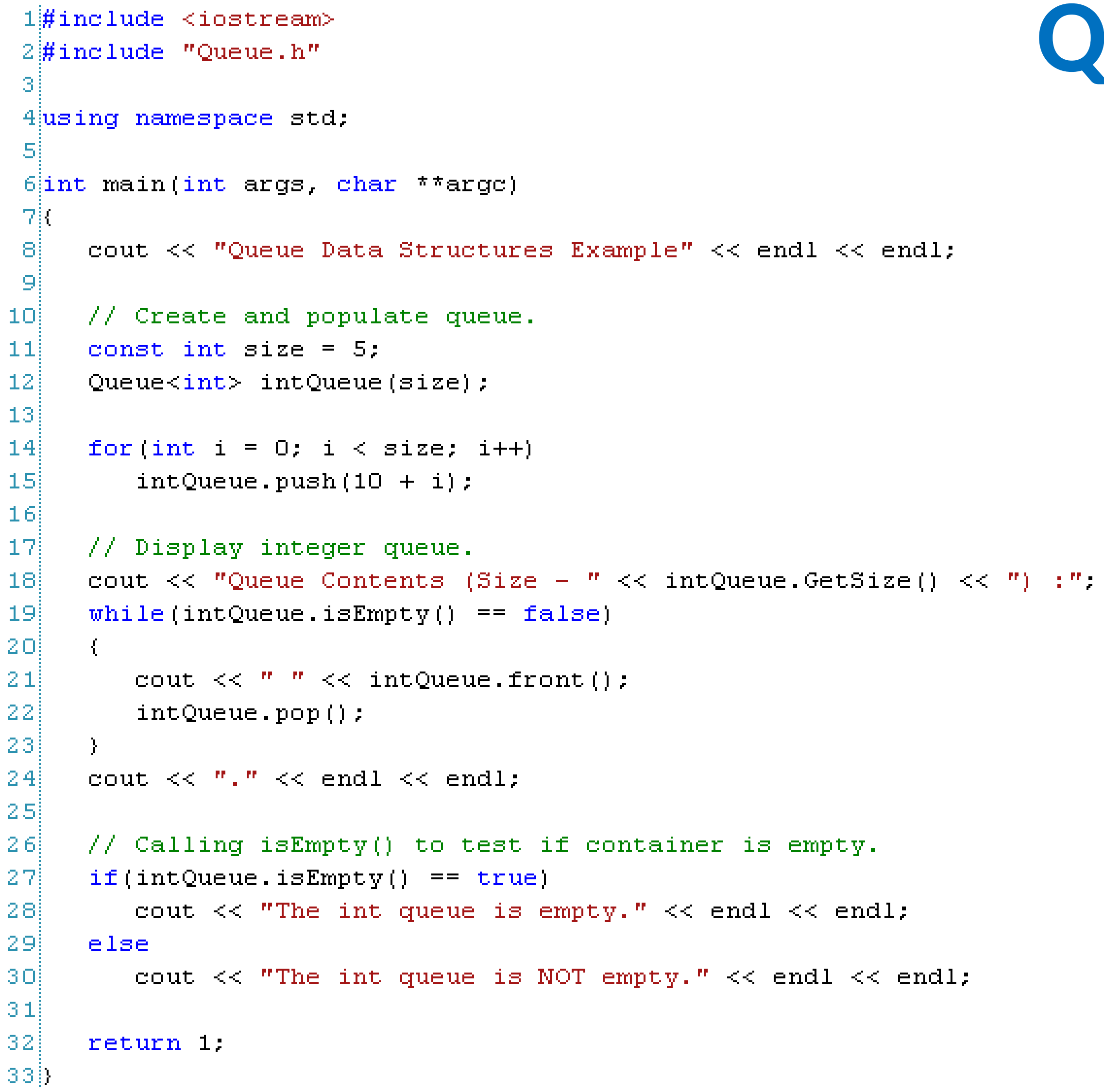
Queues

- queue
  - inserted into the front of the container and removed from the back
- double-ended queue
  - inserted and removed from the front and the back of the container
- priority queue
  - order objects based on priority
- circular queue
  - allows for the wrapping of elements

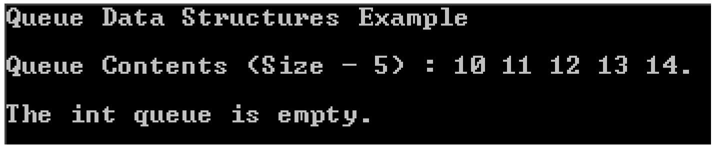
Queue

```cpp
#include "DoublyLinkList.h"

template<class T>
class Queue
{
   public:
      Queue(int size)
      {
         assert(size > 0);
         m_size = size;
      }

      ~Queue()
      {

      }

      void push(T val)
      {
         if(m_elements.GetSize() < m_size)
            m_elements.Push(val);
      }

      void pop()
      {
         m_elements.Pop_Front();
      }
      T& front()
      {
         LinkIterator<T> it;
         it = m_elements.Begin();

         return *it;
      }

      T& back()
      {
         LinkIterator<T> it;
         it = m_elements.Last();

         return *it;
      }

      int GetSize()     { return m_elements.GetSize(); }
      int GetMaxSize()  { return m_size; }
      bool isEmpty()    { return (m_elements.GetSize() == 0); }

      void Resize(int size) { assert(size > 0); m_size = size; }


   private:
      LinkList<T> m_elements;
      int m_size;
};
```

Queue Example



```cpp
#include <iostream>
#include "Queue.h"

using namespace std;

int main(int args, char **argc)
{
   cout << "Queue Data Structures Example" << endl << endl;

   // Create and populate queue.
   const int size = 5;
   Queue<int> intQueue(size);

   for(int i = 0; i < size; i++)
      intQueue.push(10 + i);

   // Display integer queue.
   cout << "Queue Contents (Size - " << intQueue.GetSize() << ") :";
   while(intQueue.isEmpty() == false)
   {
      cout << " " << intQueue.front();
      intQueue.pop();
   }
   cout << "." << endl << endl;

   // Calling isEmpty() to test if container is empty.
   if(intQueue.isEmpty() == true)
      cout << "The int queue is empty." << endl << endl;
   else
      cout << "The int queue is NOT empty." << endl << endl;

   return 1;
}
```



```
Queue Data Structures Example

Queue Contents (Size - 5) : 10 11 12 13 14.

The int queue is empty.
```

Double-Ended Queue (Deque)

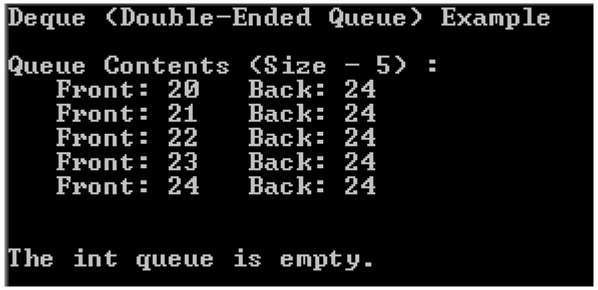
- allows for the insertion, removal, and peeking of objects from both ends of the container

Deque

```cpp
#include "DoublyLinkList.h"

template<typename T>
class Queue
{
   public:
      Queue(int size)
      {
         assert(size > 0);
         m_size = size;
      }

      ~Queue()
      {
      }

      void push_front(T val)
      {
         if(m_elements.GetSize() < m_size)
            m_elements.Push_Front(val);
      }

      void push_back(T val)
      {
         if(m_elements.GetSize() < m_size)
            m_elements.Push(val);
      }

      void pop_front()
      {
         m_elements.Pop();
      }
      void pop_back()
      {
         m_elements.Pop_Front();
      }

      T& front()
      {
         LinkIterator<T> it;
         it = m_elements.Last();

         return *it;
      }

      T& back()
      {
         LinkIterator<T> it;
         it = m_elements.Begin();

         return *it;
      }

      int GetSize()    { return m_elements.GetSize(); }
      int GetMaxSize() { return m_size; }
      bool isEmpty()   { return (m_elements.GetSize() == 0); }

      void Resize(int size) { assert(size > 0); m_size = size; }

   private:
      LinkList<T> m_elements;
      int m_size;
};
```

Deque Example

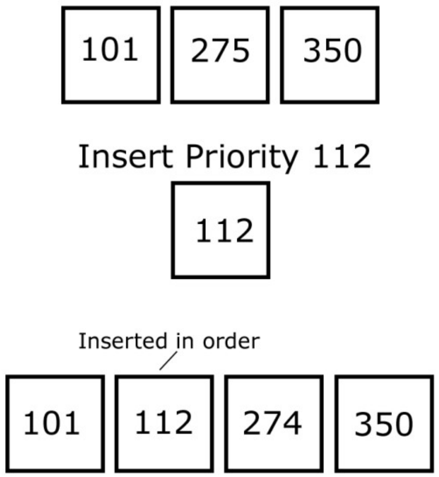
```cpp
#include <iostream>
#include "Deque.h"

using namespace std;

int main(int args, char **argc)
{
   cout << "Deque (Double-Ended Queue) Example"
        << endl << endl;

   // Create and populate queue.
   const int size = 5;
   Queue<int> intQueue(size);

   for(int i = 0; i < size; i++)
      intQueue.push_front(20 + i);

   // Display integer queue.
   cout << "Queue Contents (Size - "
        << intQueue.GetSize() << ") :" << endl;
   while(intQueue.isEmpty() == false)
   {
      cout << "   Front: " << intQueue.front();
      cout << "   Back: " << intQueue.back();
      cout << endl;

      intQueue.pop_front();
   }
   cout << endl << endl;

   // Calling isEmpty() to test if container is empty.
   if(intQueue.isEmpty() == true)
      cout << "The int queue is empty." << endl << endl;
   else
      cout << "The int queue is NOT empty." << endl << endl;

   return 1;
}
```



```
Deque (Double-Ended Queue) Example

Queue Contents (Size - 5) :
   Front: 20   Back: 24
   Front: 21   Back: 24
   Front: 22   Back: 24
   Front: 23   Back: 24
   Front: 24   Back: 24


The int queue is empty.
```

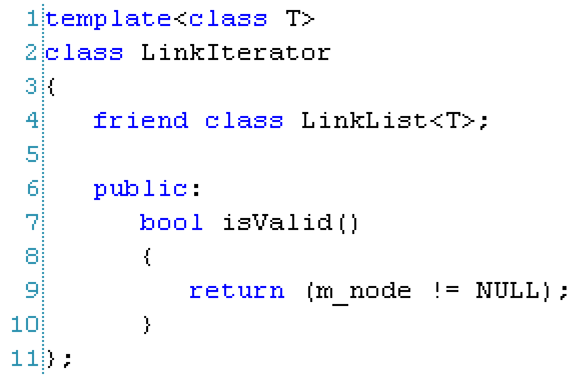
Priority Queues

- objects inserted into the data structure are ordered by importance rather than their insertion order
  - much slower big-O of 0(N)
  - STL version is faster because it uses trees



Priority Queues

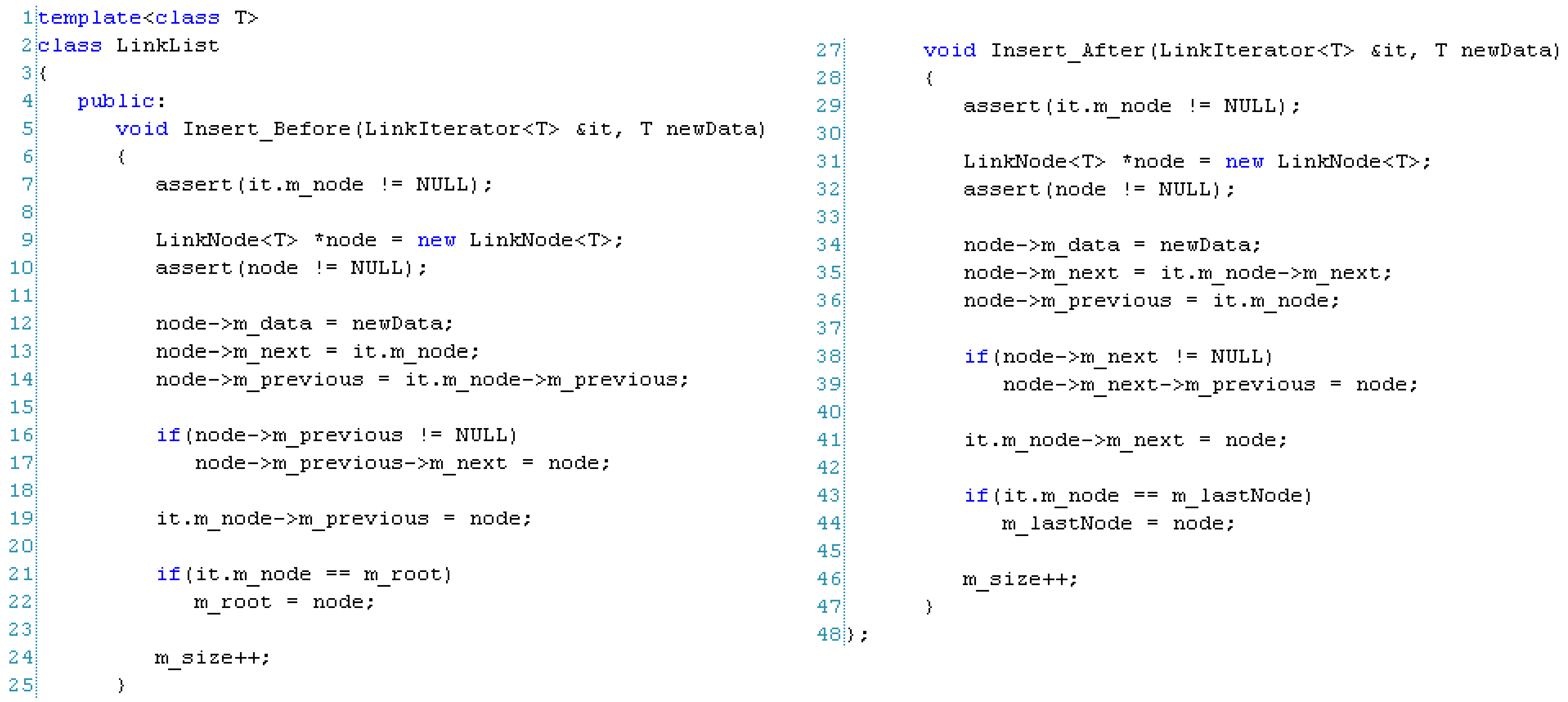
- Update in LinkIterator



```
template<class T>
class LinkIterator
{
   friend class LinkList<T>;

   public:
      bool isValid()
      {
         return (m_node != NULL);
      }
};
```

Priority Queues

- Update in LinkList



```cpp
template<class T>
class LinkList
{
   public:
      void Insert_Before(LinkIterator<T> &it, T newData)
      {
         assert(it.m_node != NULL);

         LinkNode<T> *node = new LinkNode<T>;
         assert(node != NULL);

         node->m_data = newData;
         node->m_next = it.m_node;
         node->m_previous = it.m_node->m_previous;

         if(node->m_previous != NULL)
            node->m_previous->m_next = node;

         it.m_node->m_previous = node;

         if(it.m_node == m_root)
            m_root = node;

         m_size++;
      }
      void Insert_After(LinkIterator<T> &it, T newData)
      {
         assert(it.m_node != NULL);

         LinkNode<T> *node = new LinkNode<T>;
         assert(node != NULL);

         node->m_data = newData;
         node->m_next = it.m_node->m_next;
         node->m_previous = it.m_node;

         if(node->m_next != NULL)
            node->m_next->m_previous = node;

         it.m_node->m_next = node;

         if(it.m_node == m_lastNode)
            m_lastNode = node;

         m_size++;
      }
};
```

Priority Queues

```cpp
#include "PriorityQueueLinkList.h"

template<typename T, typename CMP>
class PriorityQueue
{
   public:
      PriorityQueue(int size)
      {
         assert(size > 0);
         m_size = size;
      }

      void pop()
      {
         m_elements.Pop_Front();
      }

      T& front()
      {
         LinkIterator<T> it;
         it = m_elements.Begin();
         return *it;
      }

      T& back()
      {
         LinkIterator<T> it;
         it = m_elements.Last();
         return *it;
      }
```

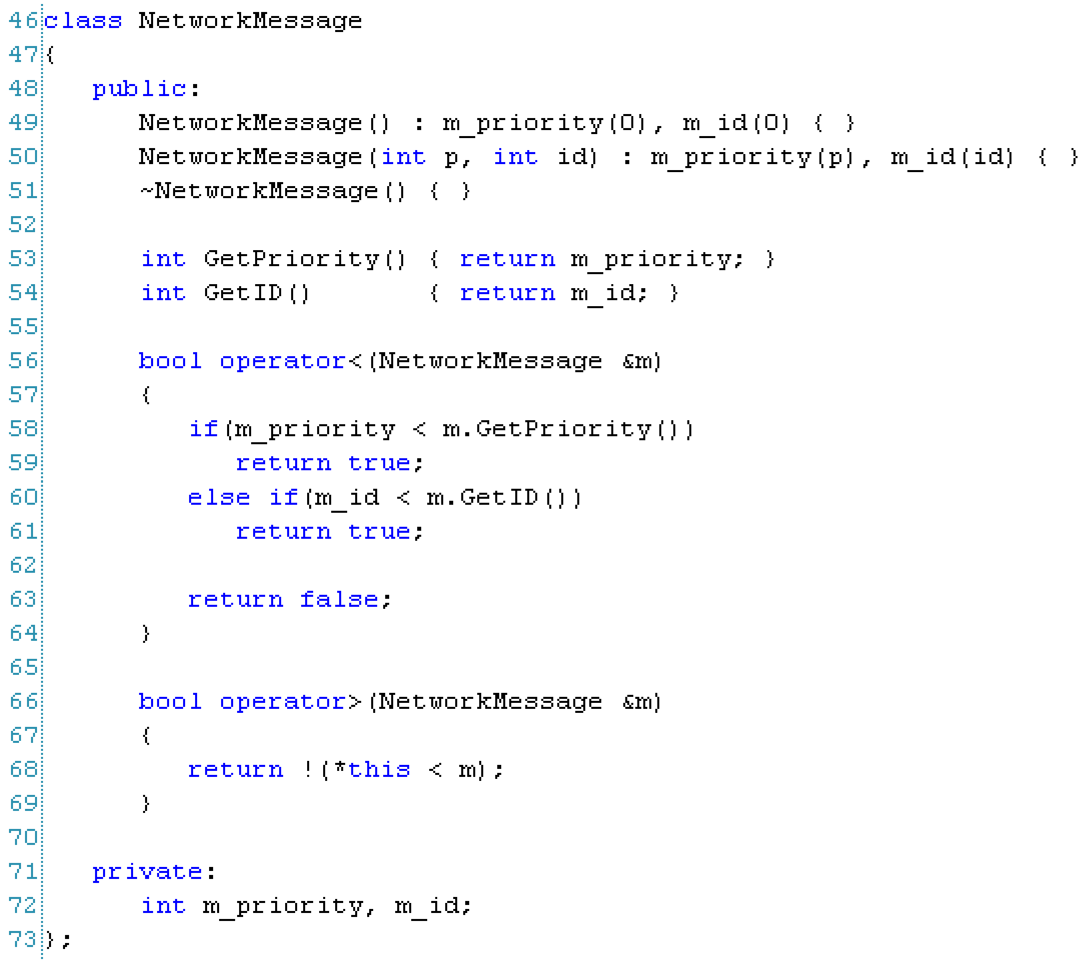
```cpp
      void push(T val)
      {
         assert(m_elements.GetSize() < m_size);
         if(m_elements.GetSize() == 0)
         {
            m_elements.Push(val);
         }
         else
         {
            LinkIterator<T> it;
            it = m_elements.Begin();

            CMP cmp;
            while(it.isValid())
            {
               if(cmp(val, *it))
                  break;
               it++;
            }

            if(it.isValid())
               m_elements.Insert_Before(it, val);
            else
               m_elements.Push(val);
         }
      }

      int GetSize()     { return m_elements.GetSize(); }
      int GetMaxSize()  { return m_size; }
      bool isEmpty()    { return (m_elements.GetSize() == 0); }
      void Resize(int size) { assert(size > 0); m_size = size; }

   private:
      LinkList<T> m_elements;
      int m_size;
};
```

Priority Queue Example

```cpp
#include <iostream>
#include "PriorityQueue.h"

using namespace std;

template<typename T>
class less_cmp
{
   public:
      inline bool operator()(T lVal, T rVal)
      {
         return (lVal < rVal);
      }
};

template<typename T>
class less_cmp_ptr
{
   public:
      inline bool operator()(T lVal, T rVal)
      {
         return ((*lVal) < (*rVal));
      }
};
```

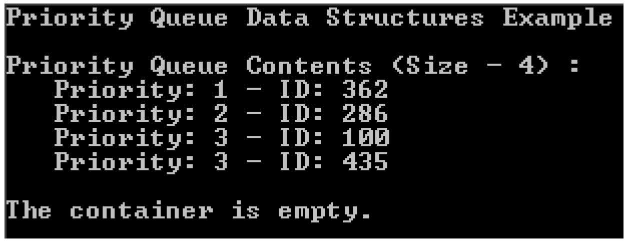
```cpp
template<typename T>
class greater_cmp
{
   public:
      inline bool operator()(T lVal, T rVal)
      {
         return !(lVal < rVal);
      }
};

template<typename T>
class greater_cmp_ptr
{
   public:
      inline bool operator()(T lVal, T rVal)
      {
         return !((*lVal) < (*rVal));
      }
};
```

Priority Queue Example



```cpp
class NetworkMessage
{
   public:
      NetworkMessage() : m_priority(0), m_id(0) { }
      NetworkMessage(int p, int id) : m_priority(p), m_id(id) { }
      ~NetworkMessage() { }

      int GetPriority() { return m_priority; }
      int GetID()       { return m_id; }

      bool operator<(NetworkMessage &m)
      {
         if(m_priority < m.GetPriority())
            return true;
         else if(m_id < m.GetID())
            return true;

         return false;
      }

      bool operator>(NetworkMessage &m)
      {
         return !(*this < m);
      }

   private:
      int m_priority, m_id;
};
```

```cpp
int main(int args, char **argc)
{
   cout << "Priority Queue Data Structures Example" << endl;
   cout << endl;

   // Create and populate queue.
   const int size = 4;
   PriorityQueue<NetworkMessage,
                 less_cmp<NetworkMessage> > que(size);

   que.push(NetworkMessage(3, 100));
   que.push(NetworkMessage(2, 286));
   que.push(NetworkMessage(1, 362));
   que.push(NetworkMessage(3, 435));

   // Display integer queue.
   cout << "Priority Queue Contents (Size - "
        << que.GetSize() << ") :" << endl;
   while(que.isEmpty() == false)
   {
      cout << "   Priority: " << que.front().GetPriority();
      cout << " - ID: "       << que.front().GetID();
      cout << endl;

      que.pop();
   }
   cout << endl;

   // Calling isEmpty() to test if container is empty.
   if(que.isEmpty() == true)
      cout << "The container is empty." << endl << endl;
   else
      cout << "The container is NOT empty." << endl << endl;

   return 1;
}
```



```
Priority Queue Data Structures Example

Priority Queue Contents (Size - 4) :
   Priority: 1 - ID: 362
   Priority: 2 - ID: 286
   Priority: 3 - ID: 100
   Priority: 3 - ID: 435

The container is empty.
```

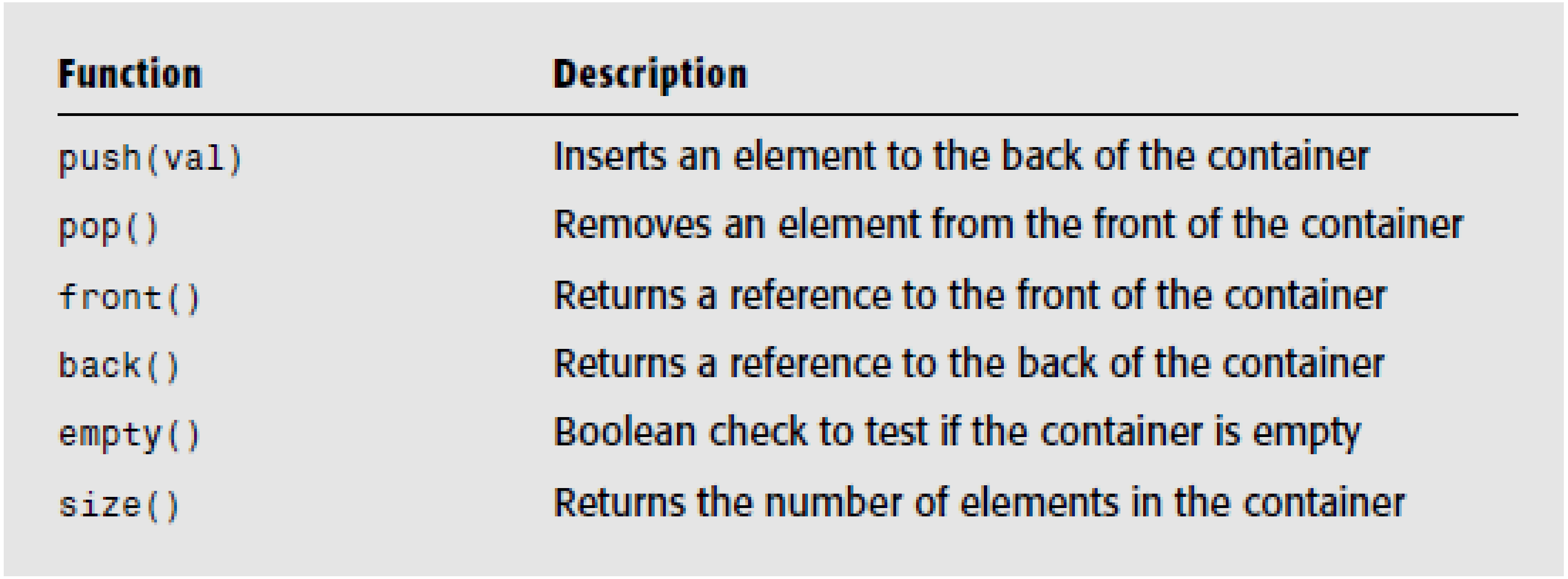
STL Queues

- STL queue
- STL deque
- STL priority_queue

STL queue

- enables insertions and removals of elements to occur at the front of the container
- by default, the queue class is implemented with a deque (better performance)
  - it can also be implemented using a list (STL link list)

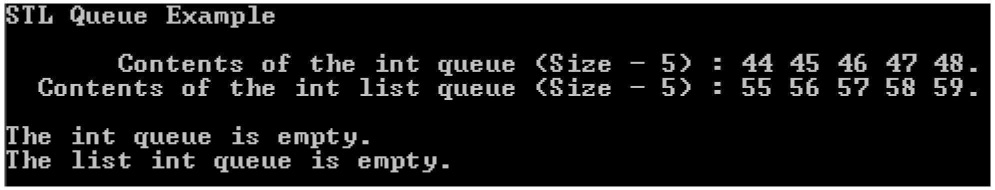
STL queue



Function	Description
`push(val)`	Inserts an element to the back of the container
`pop()`	Removes an element from the front of the container
`front()`	Returns a reference to the front of the container
`back()`	Returns a reference to the back of the container
`empty()`	Boolean check to test if the container is empty
`size()`	Returns the number of elements in the container

Example

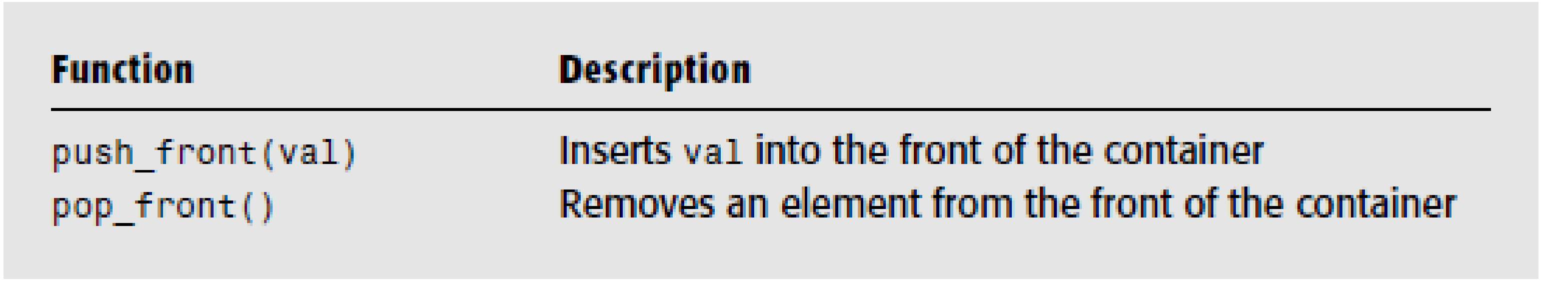
```cpp
#include <iostream>
#include <queue>
#include <list>

using namespace std;

template<typename T>
void DisplayQueue(T &que)
{
   cout << "(Size - " << que.size() << ") :";

   while(que.empty() == false)
   {
      cout << " " << que.front();
      que.pop();
   }

   cout << "." << endl;
}

int main(int args, char **argc)
{
   cout << "STL Queue Example" << endl << endl;

   queue<int> intQueue;
   queue<int, list<int> > listQueue;

   for(int i = 0; i < 5; i++)
   {
      intQueue.push(44 + i);
      listQueue.push(55 + i);
   }

   // Display normal (deque) integer queue.
   cout << "      Contents of the int queue ";
   DisplayQueue(intQueue);


   // Display link list integer queue.
   cout << " Contents of the int list queue ";
   DisplayQueue(listQueue);

   cout << endl;

   // Calling empty() to test if container is empty.
   if(intQueue.empty() == true)
      cout << "The int queue is empty." << endl;
   else
      cout << "The int queue is NOT empty." << endl;

   // Calling empty() to test if container is empty.
   if(listQueue.empty() == true)
      cout << "The list int queue is empty." << endl;
   else
      cout << "The list int queue is NOT empty." << endl;

   cout << endl;

   return 1;
}
```



```
STL Queue Example

       Contents of the int queue (Size - 5) : 44 45 46 47 48.
  Contents of the int list queue (Size - 5) : 55 56 57 58 59.

The int queue is empty.
The list int queue is empty.
```

STL deque

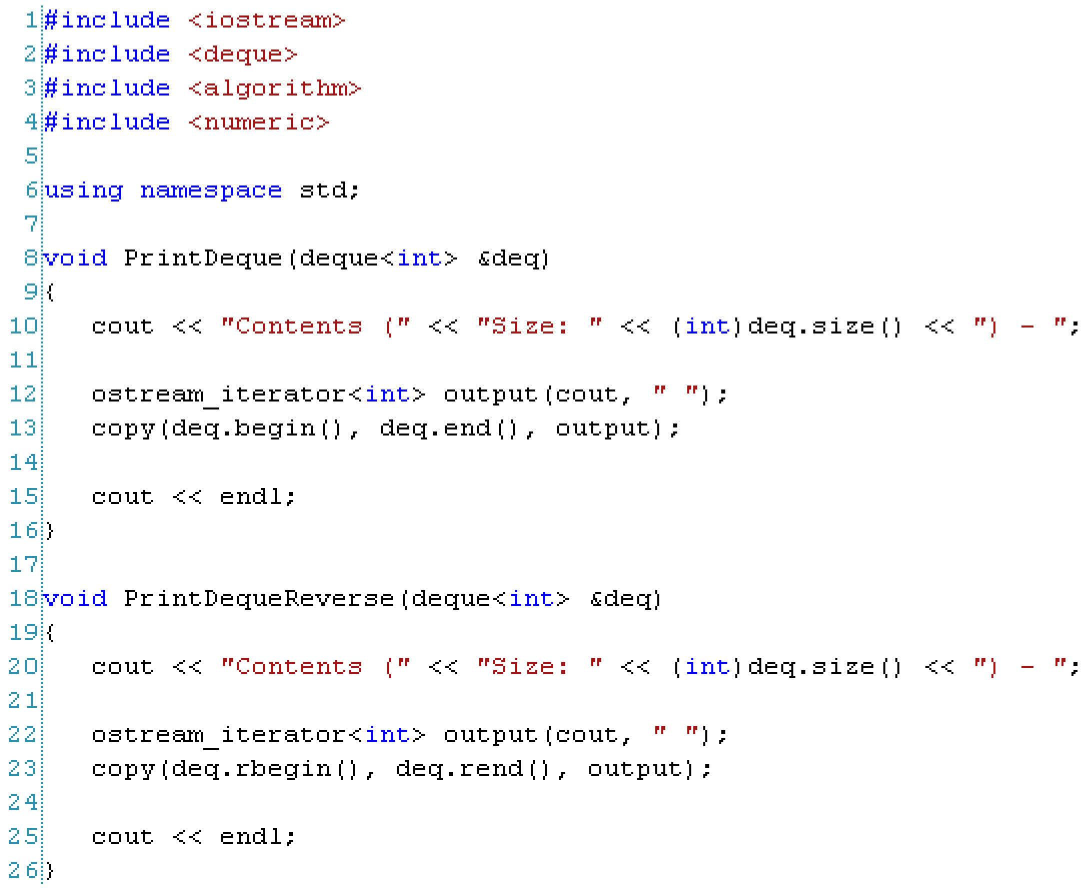
- double-ended queue container
- provides indexed access using subscripting for reading and writing elements
- has support of random-access iterators (STL vector)



Function	Description
`push_front(val)`	Inserts `val` into the front of the container
`pop_front()`	Removes an element from the front of the container

Members of the deque Template Class that Differ from the vector Class

STL deque example

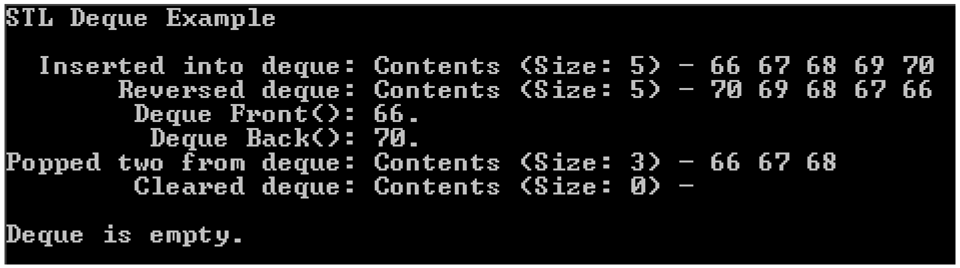


```cpp
#include <iostream>
#include <deque>
#include <algorithm>
#include <numeric>

using namespace std;

void PrintDeque(deque<int> &deq)
{
   cout << "Contents (" << "Size: " << (int)deq.size() << ") - ";

   ostream_iterator<int> output(cout, " ");
   copy(deq.begin(), deq.end(), output);

   cout << endl;
}

void PrintDequeReverse(deque<int> &deq)
{
   cout << "Contents (" << "Size: " << (int)deq.size() << ") - ";

   ostream_iterator<int> output(cout, " ");
   copy(deq.rbegin(), deq.rend(), output);

   cout << endl;
}
```

STL deque example

```cpp
int main(int args, char **argc)
{
   cout << "STL Deque Example" << endl << endl;

   deque<int> intDeque;

   for(int i = 0; i < 5; i++)
      intDeque.push_back(66 + i);

   // Display deque.
   cout << " Inserted into deque: ";
   PrintDeque(intDeque);

   cout << "        Reversed deque: ";
   PrintDequeReverse(intDeque);

   // Display item at the front of deque.
   cout << "          Deque Front(): "
        << intDeque.front() << "." << endl;

   // Display item at the front of deque.
   cout << "           Deque Back(): "
        << intDeque.back() << "." << endl;

   // Pop off the container.
   intDeque.pop_back();
   intDeque.pop_back();

   cout << "Popped two from deque: ";
   PrintDeque(intDeque);
```

```cpp
   // Clear the container.
   intDeque.clear();

   cout << "         Cleared deque: ";
   PrintDeque(intDeque);

   cout << endl;

   // Test if the container is empty.
   if(intDeque.empty() == true)
      cout << "Deque is empty.";
   else
      cout << "Deque is NOT empty.";

   cout << endl << endl;

   return 1;
}
```

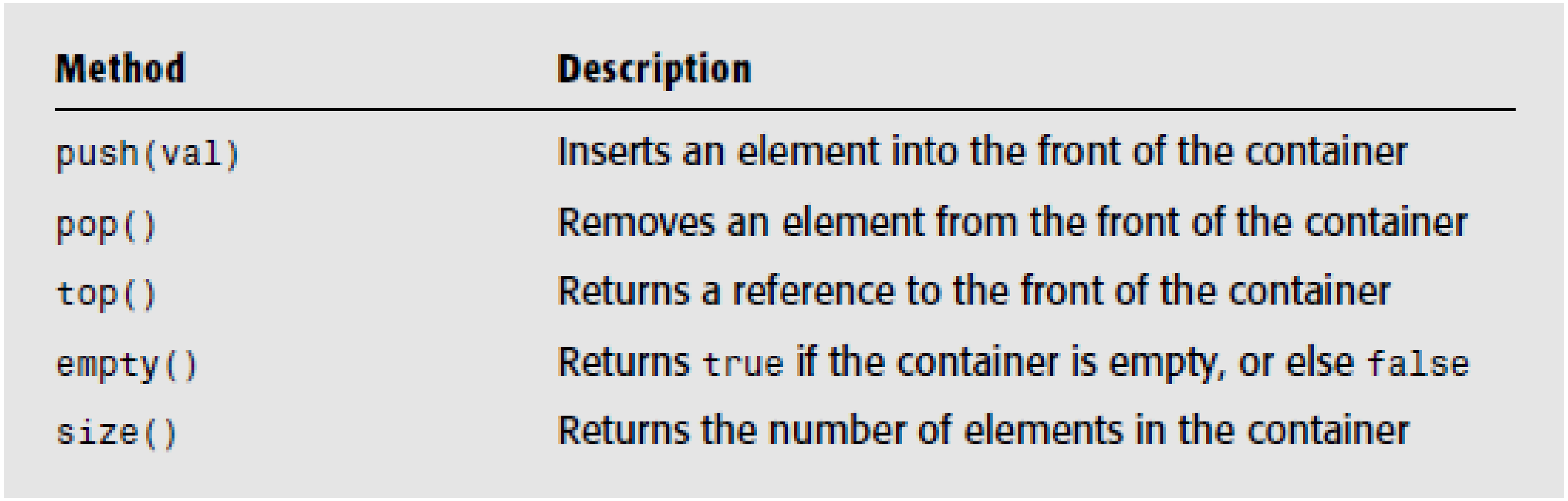


```
STL Deque Example

  Inserted into deque: Contents (Size: 5) - 66 67 68 69 70
       Reversed deque: Contents (Size: 5) - 70 69 68 67 66
        Deque Front(): 66.
         Deque Back(): 70.
Popped two from deque: Contents (Size: 3) - 66 67 68
        Cleared deque: Contents (Size: 0) -

Deque is empty.
```

STL priority_queue

- sorts elements, usually using a heap-sort
- and allows for the removal of elements from the front of the container
- by default, it uses a vector as its underlying data structure
  - it can also use a deque
- by default, it sorts elements in less-than to greater-than order
  - can be specified by using a comparison template

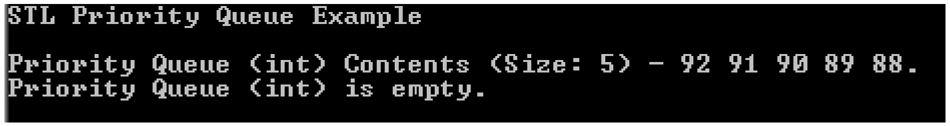
STL priority_queue



Method	Description
`push(val)`	Inserts an element into the front of the container
`pop()`	Removes an element from the front of the container
`top()`	Returns a reference to the front of the container
`empty()`	Returns `true` if the container is empty, or else `false`
`size()`	Returns the number of elements in the container

Example 1

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

using namespace std;

int main(int args, char **argc)
{
   cout << "STL Priority Queue Example" << endl << endl;

   priority_queue<int> priQueue;

   for(int i = 0; i < 5; i++)
      priQueue.push(88 + i);

   cout << "Priority Queue (int) Contents (" << "Size: "
       << (int)priQueue.size() << ") -";

   int size = (int)priQueue.size();

   for(int i = 0; i < size; i++)
   {
      cout << " " << priQueue.top();
      priQueue.pop();
   }

   cout << "." << endl;

   if(priQueue.empty() == true)
      cout << "Priority Queue (int) is empty.";
   else
      cout << "Priority Queue (int) is NOT empty.";

   cout << endl << endl;

   return 1;
}
```



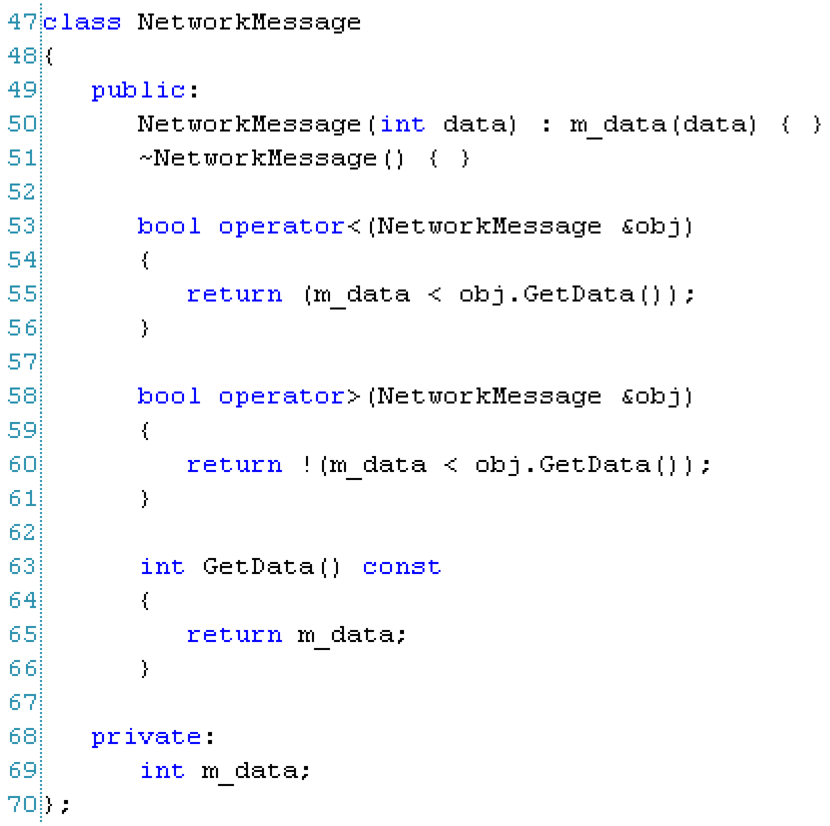
```
STL Priority Queue Example

Priority Queue (int) Contents (Size: 5) - 92 91 90 89 88.
Priority Queue (int) is empty.
```

Example 2

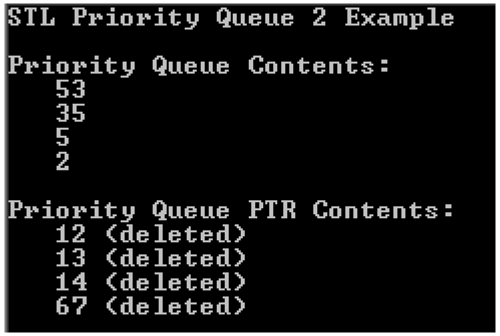
```cpp
#include <iostream>
#include <queue>
#include <vector>

using namespace std;

template<typename T>
class less_cmp
{
    public:
        inline bool operator()(T lVal, T rVal)
        {
            return (lVal < rVal);
        }
};

template<typename T>
class less_cmp_ptr
{
    public:
        inline bool operator()(T lVal, T rVal)
        {
            return ((*lVal) < (*rVal));
        }
};
```

```cpp
template<typename T>
class greater_cmp
{
    public:
        inline bool operator()(T lVal, T rVal)
        {
            return !(lVal < rVal);
        }
};

template<typename T>
class greater_cmp_ptr
{
    public:
        inline bool operator()(T lVal, T rVal)
        {
            return !((*lVal) < (*rVal));
        }
};
```

Example 2



```cpp
class NetworkMessage
{
   public:
      NetworkMessage(int data) : m_data(data) { }
      ~NetworkMessage() { }

      bool operator<(NetworkMessage &obj)
      {
         return (m_data < obj.GetData());
      }

      bool operator>(NetworkMessage &obj)
      {
         return !(m_data < obj.GetData());
      }

      int GetData() const
      {
         return m_data;
      }

   private:
      int m_data;
};
```



```
STL Priority Queue 2 Example

Priority Queue Contents:
   53
   35
   5
   2

Priority Queue PTR Contents:
   12 (deleted)
   13 (deleted)
   14 (deleted)
   67 (deleted)
```

```cpp
int main(int args, char **argc)
{
   cout << "STL Priority Queue 2 Example" << endl << endl;

   // Create two test priority queues.
   priority_queue<NetworkMessage, vector<NetworkMessage>,
                  less_cmp<NetworkMessage> > priQueue;

   priority_queue<NetworkMessage*, vector<NetworkMessage*>,
                  greater_cmp_ptr<NetworkMessage*> > priQueuePtr;

   priQueue.push(NetworkMessage(5));
   priQueue.push(NetworkMessage(35));
   priQueue.push(NetworkMessage(2));
   priQueue.push(NetworkMessage(53));

   priQueuePtr.push(new NetworkMessage(14));
   priQueuePtr.push(new NetworkMessage(67));
   priQueuePtr.push(new NetworkMessage(13));
   priQueuePtr.push(new NetworkMessage(12));

   // Display priority queue.
   cout << "Priority Queue Contents:" << endl;

   int size = (int)priQueue.size();

   for(int i = 0; i < size; i++)
   {
      cout << "   " << priQueue.top().GetData() << endl;
      priQueue.pop();
   }
   cout << endl;
```

```cpp
   // Display priority queue ptr.
   cout << "Priority Queue PTR Contents:" << endl;

   size = (int)priQueuePtr.size();

   for(int i = 0; i < size; i++)
   {
      NetworkMessage *ptr = priQueuePtr.top();

      if(ptr != NULL)
      {
         cout << "   " << ptr->GetData();
         delete ptr;
         cout << " (deleted)" << endl;
      }

      priQueuePtr.pop();
   }

   cout << endl << endl;

   return 1;
}
```