

GAME2001 Data Structures and Algorithms

Fall 2020

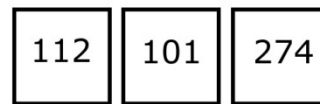


Week 9

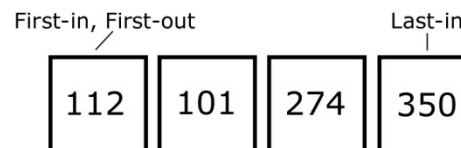
Stacks and Queues II

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



Insert -> 350 <- Element



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

```
1 #include "DoublyLinkedList.h"
2
3 template<class T>
4 class Queue
5 {
6     public:
7         Queue(int size)
8         {
9             assert(size > 0);
10            m_size = size;
11        }
12
13        ~Queue()
14        {
15        }
16
17        void push(T val)
18        {
19            if(m_elements.GetSize() < m_size)
20                m_elements.Push(val);
21        }
22
23        void pop()
24        {
25            m_elements.Pop_Front();
26        }
27    }
```

```
29     T& front()
30     {
31         LinkIterator<T> it;
32         it = m_elements.Begin();
33
34         return *it;
35     }
36
37     T& back()
38     {
39         LinkIterator<T> it;
40         it = m_elements.Last();
41
42         return *it;
43     }
44
45     int GetSize() { return m_elements.GetSize(); }
46     int GetMaxSize() { return m_size; }
47     bool isEmpty() { return (m_elements.GetSize() == 0); }
48
49     void Resize(int size) { assert(size > 0); m_size = size; }
50
51     private:
52         LinkList<T> m_elements;
53         int m_size;
54     };
55 }
```

Queue Example

```

1#include <iostream>
2#include "Queue.h"
3
4using namespace std;
5
6int main(int args, char **argc)
7{
8    cout << "Queue Data Structures Example" << endl << endl;
9
10    // Create and populate queue.
11    const int size = 5;
12    Queue<int> intQueue(size);
13
14    for(int i = 0; i < size; i++)
15        intQueue.push(10 + i);
16
17    // Display integer queue.
18    cout << "Queue Contents (Size - " << intQueue.GetSize() << " ) :";
19    while(intQueue.isEmpty() == false)
20    {
21        cout << " " << intQueue.front();
22        intQueue.pop();
23    }
24    cout << "." << endl << endl;
25
26    // Calling isEmpty() to test if container is empty.
27    if(intQueue.isEmpty() == true)
28        cout << "The int queue is empty." << endl << endl;
29    else
30        cout << "The int queue is NOT empty." << endl << endl;
31
32    return 1;
33}

```

```

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

```

1#include "DoublyLinkedList.h"
2
3template<typename T>
4class Queue
5{
6    public:
7        Queue(int size)
8        {
9            assert(size > 0);
10           m_size = size;
11        }
12
13        ~Queue()
14        {
15        }
16
17        void push_front(T val)
18        {
19            if(m_elements.GetSize() < m_size)
20                m_elements.Push_Front(val);
21        }
22
23        void push_back(T val)
24        {
25            if(m_elements.GetSize() < m_size)
26                m_elements.Push(val);
27        }
28
29        void pop_front()
30        {
31            m_elements.Pop();
32        }
33
34        void pop_back()
35        {
36            m_elements.Pop_Back();
37        }
38
39        T& front()
40        {
41            LinkIterator<T> it;
42            it = m_elements.Last();
43
44            return *it;
45        }
46
47        T& back()
48        {
49            LinkIterator<T> it;
50            it = m_elements.Begin();
51
52            return *it;
53        }
54
55        int GetSize() { return m_elements.GetSize(); }
56        int GetMaxSize() { return m_size; }
57        bool isEmpty() { return (m_elements.GetSize() == 0); }
58
59        void Resize(int size) { assert(size > 0); m_size = size; }
60
61    private:
62        LinkList<T> m_elements;
63        int m_size;
64};

```


Deque Example

```
1#include <iostream>
2#include "Deque.h"
3
4using namespace std;
5
6int main(int args, char **argc)
7{
8    cout << "Deque (Double-Ended Queue) Example"
9        << endl << endl;
10
11    // Create and populate queue.
12    const int size = 5;
13    Queue<int> intQueue(size);
14
15    for(int i = 0; i < size; i++)
16        intQueue.push_front(20 + i);
17
18    // Display integer queue.
19    cout << "Queue Contents (Size - "
20        << intQueue.GetSize() << ") :" << endl;
21    while(intQueue.isEmpty() == false)
22    {
23        cout << "    Front: " << intQueue.front();
24        cout << "    Back: " << intQueue.back();
25        cout << endl;
26
27        intQueue.pop_front();
28    }
29    cout << endl << endl;
```

```
31    // Calling isEmpty() to test if container is empty.
32    if(intQueue.isEmpty() == true)
33        cout << "The int queue is empty." << endl << endl;
34    else
35        cout << "The int queue is NOT empty." << endl << endl;
36
37    return 1;
38}
```

Deque (Double-Ended Queue) Example

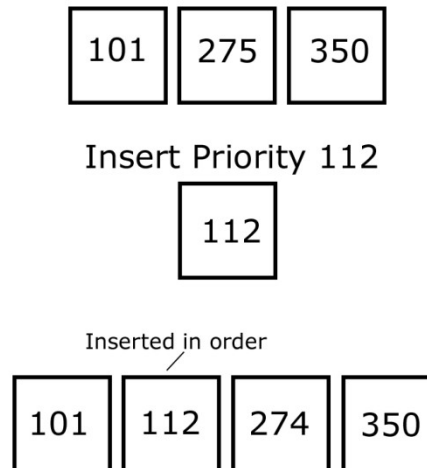
Queue Contents (Size - 5) :

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 $O(N)$
 - STL version is faster because it uses trees



Priority Queues

- Update in LinkIterator

```
1 template<class T>
2 class LinkIterator
3 {
4     friend class LinkList<T>;
5
6     public:
7         bool isValid()
8         {
9             return (m_node != NULL);
10        }
11};
```

Priority Queues

- Update in LinkList

```
1 template<class T>
2 class LinkList
3 {
4     public:
5         void Insert_Before(LinkIterator<T> &it, T newData)
6         {
7             assert(it.m_node != NULL);
8
9             LinkNode<T> *node = new LinkNode<T>;
10            assert(node != NULL);
11
12            node->m_data = newData;
13            node->m_next = it.m_node;
14            node->m_previous = it.m_node->m_previous;
15
16            if(node->m_previous != NULL)
17                node->m_previous->m_next = node;
18
19            it.m_node->m_previous = node;
20
21            if(it.m_node == m_root)
22                m_root = node;
23
24            m_size++;
25        }
```

```
27     void Insert_After(LinkIterator<T> &it, T newData)
28     {
29         assert(it.m_node != NULL);
30
31         LinkNode<T> *node = new LinkNode<T>;
32         assert(node != NULL);
33
34         node->m_data = newData;
35         node->m_next = it.m_node->m_next;
36         node->m_previous = it.m_node;
37
38         if(node->m_next != NULL)
39             node->m_next->m_previous = node;
40
41         it.m_node->m_next = node;
42
43         if(it.m_node == m_lastNode)
44             m_lastNode = node;
45
46         m_size++;
47     }
48 }
```

Priority Queues

```
1#include "PriorityQueueLinkedList.h"
2
3template<typename T, typename CMP>
4class PriorityQueue
5{
6    public:
7        PriorityQueue(int size)
8        {
9            assert(size > 0);
10           m_size = size;
11        }
12
13        void pop()
14        {
15            m_elements.Pop_Front();
16        }
17
18        T& front()
19        {
20            LinkIterator<T> it;
21            it = m_elements.Begin();
22            return *it;
23        }
24
25        T& back()
26        {
27            LinkIterator<T> it;
28            it = m_elements.Last();
29            return *it;
30        }
```

```
32
33
34        void push(T val)
35        {
36            assert(m_elements.GetSize() < m_size);
37            if(m_elements.GetSize() == 0)
38            {
39                m_elements.Push(val);
40            }
41            else
42            {
43                LinkIterator<T> it;
44                it = m_elements.Begin();
45
46                CMP cmp;
47                while(it.isValid())
48                {
49                    if(cmp(val, *it))
50                        break;
51                    it++;
52                }
53
54                if(it.isValid())
55                    m_elements.Insert_Before(it, val);
56                else
57                    m_elements.Push(val);
58            }
59        }
60
61        int GetSize() { return m_elements.GetSize(); }
62        int GetMaxSize() { return m_size; }
63        bool isEmpty() { return (m_elements.GetSize() == 0); }
64        void Resize(int size) { assert(size > 0); m_size = size; }
65
66    private:
67        LinkList<T> m_elements;
68        int m_size;
69    };
67};
```

Priority Queue Example

```
1#include <iostream>
2#include "PriorityQueue.h"
3
4using namespace std;
5
6template<typename T>
7class less_cmp
8{
9    public:
10        inline bool operator()(T lVal, T rVal)
11        {
12            return (lVal < rVal);
13        }
14};
15
16template<typename T>
17class less_cmp_ptr
18{
19    public:
20        inline bool operator()(T lVal, T rVal)
21        {
22            return ((*lVal) < (*rVal));
23        }
24};
```

```
26template<typename T>
27class greater_cmp
28{
29    public:
30        inline bool operator()(T lVal, T rVal)
31        {
32            return !(lVal < rVal);
33        }
34};
35
36template<typename T>
37class greater_cmp_ptr
38{
39    public:
40        inline bool operator()(T lVal, T rVal)
41        {
42            return !((*lVal) < (*rVal));
43        }
44};
```

Priority Queue Example

```
46 class NetworkMessage
47 {
48     public:
49         NetworkMessage() : m_priority(0), m_id(0) { }
50         NetworkMessage(int p, int id) : m_priority(p), m_id(id) { }
51         ~NetworkMessage() { }
52
53         int GetPriority() { return m_priority; }
54         int GetID()      { return m_id; }
55
56         bool operator<(NetworkMessage &m)
57         {
58             if(m_priority < m.GetPriority())
59                 return true;
60             else if(m_id < m.GetID())
61                 return true;
62
63             return false;
64         }
65
66         bool operator>(NetworkMessage &m)
67         {
68             return !(*this < m);
69         }
70
71     private:
72         int m_priority, m_id;
73 };
```

GAME 2001

Data Structures and Algorithms



```
76 int main(int argc, char **argv)
77 {
78     cout << "Priority Queue Data Structures Example" << endl;
79     cout << endl;
80
81     // Create and populate queue.
82     const int size = 4;
83     PriorityQueue<NetworkMessage,
84                 less_cmp<NetworkMessage> > que(size);
85
86     que.push(NetworkMessage(3, 100));
87     que.push(NetworkMessage(2, 286));
88     que.push(NetworkMessage(1, 362));
89     que.push(NetworkMessage(3, 435));
90
91     // Display integer queue.
92     cout << "Priority Queue Contents (Size - "
93          << que.GetSize() << ") : " << endl;
94     while(que.isEmpty() == false)
95     {
96         cout << "    Priority: " << que.front().GetPriority();
97         cout << " - ID: " << que.front().GetID();
98         cout << endl;
99
100        que.pop();
101    }
102    cout << endl;
103
104    // Calling isEmpty() to test if container is empty.
105    if(que.isEmpty() == true)
106        cout << "The container is empty." << endl << endl;
107    else
108        cout << "The container is NOT empty." << endl << endl;
109
110    return 1;
111 }
```

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

Example

```
1#include <iostream>
2#include <queue>
3#include <list>
4
5using namespace std;
6
7template<typename T>
8void DisplayQueue(T &que)
9{
10    cout << "(Size - " << que.size() << " ) :";
11
12    while(que.empty() == false)
13    {
14        cout << " " << que.front();
15        que.pop();
16    }
17
18    cout << "." << endl;
19}
20
21int main(int args, char **args)
22{
23    cout << "STL Queue Example" << endl << endl;
24
25    queue<int> intQueue;
26    queue<int, list<int> > listQueue;
27
28    for(int i = 0; i < 5; i++)
29    {
30        intQueue.push(44 + i);
31        listQueue.push(55 + i);
32    }
```

```
34    // Display normal (deque) integer queue.
35    cout << "        Contents of the int queue ";
36    DisplayQueue(intQueue);
37
38
39    // Display link list integer queue.
40    cout << "        Contents of the int list queue ";
41    DisplayQueue(listQueue);
42
43    cout << endl;
44
45    // Calling empty() to test if container is empty.
46    if(intQueue.empty() == true)
47        cout << "The int queue is empty." << endl;
48    else
49        cout << "The int queue is NOT empty." << endl;
50
51    // Calling empty() to test if container is empty.
52    if(listQueue.empty() == true)
53        cout << "The list int queue is empty." << endl;
54    else
55        cout << "The list int queue is NOT empty." << endl;
56
57    cout << endl;
58
59    return 1;
60}
```

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
<code>push_front(val)</code>	Inserts <code>val</code> into the front of the container
<code>pop_front()</code>	Removes an element from the front of the container

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

STL deque example

```
1#include <iostream>
2#include <deque>
3#include <algorithm>
4#include <numeric>
5
6using namespace std;
7
8void PrintDeque(deque<int> &deq)
9{
10    cout << "Contents (" << "Size: " << (int)deq.size() << ") - ";
11
12    ostream_iterator<int> output(cout, " ");
13    copy(deq.begin(), deq.end(), output);
14
15    cout << endl;
16}
17
18void PrintDequeReverse(deque<int> &deq)
19{
20    cout << "Contents (" << "Size: " << (int)deq.size() << ") - ";
21
22    ostream_iterator<int> output(cout, " ");
23    copy(deq.rbegin(), deq.rend(), output);
24
25    cout << endl;
26}
```

STL deque example

```
28 int main(int args, char **argc)
29 {
30     cout << "STL Deque Example" << endl << endl;
31
32     deque<int> intDeque;
33
34     for(int i = 0; i < 5; i++)
35         intDeque.push_back(66 + i);
36
37     // Display deque.
38     cout << "    Inserted into deque: ";
39     PrintDeque(intDeque);
40
41     cout << "    Reversed deque: ";
42     PrintDequeReverse(intDeque);
43
44     // Display item at the front of deque.
45     cout << "    Deque Front(): "
46          << intDeque.front() << "." << endl;
47
48     // Display item at the front of deque.
49     cout << "    Deque Back(): "
50          << intDeque.back() << "." << endl;
51
52     // Pop off the container.
53     intDeque.pop_back();
54     intDeque.pop_back();
55
56     cout << "Popped two from deque: ";
57     PrintDeque(intDeque);
```

```
59     // Clear the container.
60     intDeque.clear();
61
62     cout << "    Cleared deque: ";
63     PrintDeque(intDeque);
64
65     cout << endl;
66
67     // Test if the container is empty.
68     if(intDeque.empty() == true)
69         cout << "Deque is empty.";
70     else
71         cout << "Deque is NOT empty.";
72
73     cout << endl << endl;
74
75     return 1;
76 }
```

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

Example 1

```

1#include <iostream>
2#include <queue>
3
4using namespace std;
5
6int main(int args, char **argc)
7{
8    cout << "STL Priority Queue Example" << endl << endl;
9
10    priority_queue<int> priQueue;
11
12    for(int i = 0; i < 5; i++)
13        priQueue.push(88 + i);
14
15    cout << "Priority Queue (int) Contents (" << "Size: "
16        << (int)priQueue.size() << ") -";
17
18    int size = (int)priQueue.size();
19
20    for(int i = 0; i < size; i++)
21    {
22        cout << " " << priQueue.top();
23        priQueue.pop();
24    }
25
26    cout << "." << endl;
27
28    if(priQueue.empty() == true)
29        cout << "Priority Queue (int) is empty.";
30    else
31        cout << "Priority Queue (int) is NOT empty.";
32
33    cout << endl << endl;
34
35    return 1;
36}

```

STL Priority Queue Example

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

Example 2

```
1#include <iostream>
2#include <queue>
3#include <vector>
4
5using namespace std;
6
7template<typename T>
8class less_cmp
9{
10    public:
11        inline bool operator()(T lVal, T rVal)
12        {
13            return (lVal < rVal);
14        }
15};
16
17template<typename T>
18class less_cmp_ptr
19{
20    public:
21        inline bool operator()(T lVal, T rVal)
22        {
23            return ((*lVal) < (*rVal));
24        }
25};
```

```
27template<typename T>
28class greater_cmp
29{
30    public:
31        inline bool operator()(T lVal, T rVal)
32        {
33            return !(lVal < rVal);
34        }
35};
36
37template<typename T>
38class greater_cmp_ptr
39{
40    public:
41        inline bool operator()(T lVal, T rVal)
42        {
43            return !((*lVal) < (*rVal));
44        }
45};
```

Example 2

```
47 class NetworkMessage
48 {
49     public:
50         NetworkMessage(int data) : m_data(data) { }
51         ~NetworkMessage() { }
52
53         bool operator<(NetworkMessage &obj)
54         {
55             return (m_data < obj.GetData());
56         }
57
58         bool operator>(NetworkMessage &obj)
59         {
60             return !(m_data < obj.GetData());
61         }
62
63         int GetData() const
64         {
65             return m_data;
66         }
67
68     private:
69         int m_data;
70 };
```

GAME 2001

Data Structures and Algorithms

```

72 int main(int argc, char **argv)
73 {
74     cout << "STL Priority Queue 2 Example" << endl << endl;
75
76     // Create two test priority queues.
77     priority_queue<NetworkMessage, vector<NetworkMessage>,
78         less_cmp<NetworkMessage> > priQueue;
79
80     priority_queue<NetworkMessage*, vector<NetworkMessage*>,
81         greater_cmp_ptr<NetworkMessage*> > priQueuePtr;
82
83     priQueue.push(NetworkMessage(5));
84     priQueue.push(NetworkMessage(35));
85     priQueue.push(NetworkMessage(2));
86     priQueue.push(NetworkMessage(53));
87
88     priQueuePtr.push(new NetworkMessage(14));
89     priQueuePtr.push(new NetworkMessage(67));
90     priQueuePtr.push(new NetworkMessage(13));
91     priQueuePtr.push(new NetworkMessage(12));
92
93     // Display priority queue.
94     cout << "Priority Queue Contents:" << endl;
95
96     int size = (int)priQueue.size();
97
98     for(int i = 0; i < size; i++)
99     {
100         cout << "    " << priQueue.top().GetData() << endl;
101         priQueue.pop();
102     }
103     cout << endl;

```

```

105 // Display priority queue ptr.
106 cout << "Priority Queue PTR Contents:" << endl;
107
108 size = (int)priQueuePtr.size();
109
110 for(int i = 0; i < size; i++)
111 {
112     NetworkMessage *ptr = priQueuePtr.top();
113
114     if(ptr != NULL)
115     {
116         cout << "    " << ptr->GetData();
117         delete ptr;
118         cout << " (deleted)" << endl;
119     }
120
121     priQueuePtr.pop();
122 }
123
124 cout << endl << endl;
125
126 return 1;
127 }

```

STL Priority Queue 2 Example

Priority Queue Contents:

53
35
5
2

Priority Queue PTR Contents:

12 (deleted)
13 (deleted)
14 (deleted)
67 (deleted)