```
1   // author Dr Y and CPSC 122 Fall 2018
2   // date: November 14, 2018
3   // file queue.h
4
5   //   implementation for ADT Queue
6   //   data object: a queue which is a First In First Out List
7   //   data structure: a circularly linked list
8   //   operations: create, destroy, check if empty, copy
9   //               (finish)
10
11  #include"queue.h"                          // member function should have const??
12  #include <iostream>
13
14  using namespace std;
15
16  struct Node
17  {
18      ItemType item;
19      Node* next;
20  };
21
22
23     //creates an empty queue
24     //post : an empty queue
25     //usage: Queue q;
26     Queue::Queue()
27     {
28         backptr = nullptr;// set queue to empty before operator=
29     }
30
31     //copies an existing queue
32     //pre  : rhsq exists
33     //post : queue object is a copy of rhsq
34     //usage: Queue q(anotherq); or passing a Que object by value
35     ostream& operator<<(ostream& output, const Queue& rhsq)
36     {
37         Node* currentptr;
38
39         if(not rhsq.isEmpty())
40         {
41            currentptr = rhsq.backptr -> next;
42            output << "front -> ";
43            while (currentptr != rhsq.backptr)
44            {
45                output << currentptr -> item << " , ";
46                currentptr = currentptr -> next;
47            }
48            output << currentptr -> item << " <- back" << endl;
49         } else {
50             output << "the queue is empty" << endl;
51         }
52         return output;
53     }
54
55     //destroys a queue
56     //pre  : queue object exists
57     //post : queue object does not exist
58     //usage: automatically done at the end of scope
59     Queue::~Queue()
60     { // must free all nodes
61     // once getServed is working then you may use it to delete all nodes
62         bool isNotEmpty;
63         while(!isEmpty())
64         {
65             getServed(isNotEmpty);
66         }
67     }
68
69     //finds the size of a queue object
```

```
70      //pre  : queue object exists
71      //post : returns the size of the queue object
72      //usage: cout << q.getSize();
73      int Queue::getSize() const
74      {
75          // look at the two loops we have written so far
76          int size = 0;
77          Node* currentptr;
78
79          if (backptr == nullptr)
80          {
81              return size;
82          } else {
83              currentptr = backptr;
84              size++;
85              while (currentptr -> next != backptr){
86                  currentptr = currentptr -> next;
87                  size++;
88              }
89          }
90          return size;
91      }
92
93      //checks to see if a queue object is empty
94      //pre  : queue object exists
95      //post : if queue is empty returns true else returns false
96      //usage: if (q.isEmpty())
97      bool Queue::isEmpty() const
98      {
99           return backptr == nullptr;
100     }
101
102     //inserts a new item at the rear of the queue
103     //pre  : newItem has an assigned value; queue exists
104     //post : if queue object is not full, newItem is added
105     //       at the rear of the queue and isNotFull is true else isNotFull is false
106     // usage: myq.lineUp(hunter, isNotFull);
107     void Queue::lineUp(ItemType newItem, bool& isNotFull)
108     {
109         Node* newptr;
110         if(isEmpty())
111         {
112              backptr = new Node;
113              if(backptr != nullptr)
114              {
115                  backptr -> item = newItem; // add them to the back
116                  backptr -> next = backptr; // circular
117                  isNotFull = true;
118              } else {
119                  isNotFull = false;
120              }
121         } else {
122             newptr = new Node;
123             if(newptr != nullptr)
124             {
125                 newptr -> item = newItem; // store the item
126                 newptr -> next = backptr -> next; // connect the newptr to the backptr
127                 backptr -> next = newptr; // conncect the backptr to the newptr
128                 backptr = newptr; // finish the circle
129                 isNotFull = true;
130             } else {
131                 isNotFull = false;
132             }
133         }
134     }
135
136     //deletes item from the front of the queue after copying it
137     //pre  : queue exists
138     //post : if queue is nonempty, front of queue has been removed
```

```
139     //         and isNotEmpty is true else isNotEmpty is false
140     // usage: queue.getServed(isNoteEmpty);
141     void Queue::getServed(bool& isNotEmpty)
142     {
143         Node* currentptr;
144         if(not isEmpty() and backptr == backptr -> next)
145         {
146             delete backptr -> next;
147             backptr = nullptr;
148             isNotEmpty = false;
149         } else if (not isEmpty()) {
150             currentptr = new Node;
151             currentptr = backptr -> next -> next;
152             delete backptr -> next;
153             backptr -> next = currentptr;
154             isNotEmpty = true;
155         }
156     }
157
158     //copies the front item
159     //pre  : queue exists and is not empty
160     //post : the front item in the queue is copied into frontItem
161     ItemType Queue::getWhoIsServed() const
162     {
163         return backptr -> next -> item;
164     }
165
166     //copies the queue
167     //pre  : rhsq exists. queue object exists but may be empty
168     //post : queue object is a copy of rhsq
169     //usage: copyq.operator=(rhsq);
170     //      or        copyq = rhsq;
171     Queue& Queue::operator=(const Queue& rhsq)
172     {
173          Node* currentptr;
174          bool isNotFull;
175         // must protect from spike
176      if(this != &rhsq) // if these two are the same then I want to do all the stuff we
        have to do
177                          // prevents any work for spike = spike in client
178      {
179         while(not isEmpty())
180         {
181         getServed(isNotFull);
182         }
183         if(not rhsq.isEmpty())
184         {
185            currentptr = rhsq.backptr -> next;
186            while (currentptr != rhsq.backptr)
187            {
188                lineUp(currentptr -> item, isNotFull);
189                currentptr = currentptr -> next;
190            }
191            lineUp(currentptr -> item, isNotFull);
192         }
193      }
194         return *this;
195     }
196
```