```
1  // Example illusrating member functions
2  // ECE3090
3  // George F. Riley, Georgia Tech, Spring 2009
4
5  #include <iostream>
6
7  using namespace std;  // We will discuss namespaces later
8
9  // Define an arbitrary fixed size limit for queues
10 #define MAX_QUEUE_SIZE 100
11
12 class LIFOQueue
13 { // Define a last-in-first-out queue with fixed size
14 public:
15   LIFOQueue();            // Constructor
16   // Define member functions to manage the elements of the queue
17   // Note the use of the "const" keyword after some of the member function
18   // declarations.  This indicates that the member function promises
19   // not to change any of the member variables.
20   bool Enque(int);            // Enque a new integer element
21   int  Deque();               // Remove an element
22   unsigned Length() const;    // Return the number of elements in the queue
23   bool Empty() const;         // Returns true if the queue is empty
24   void Clear();               // Remove all elements in the queue
25   void Print() const;         // Print the queue contents
26   // Does it make sense to define an addition operator for LIFOQueue objects?
27   // We will define one using "member function operators"
28   // Note we don't need the "left hand side" argument. Why?
29   LIFOQueue operator+(const LIFOQueue& rhs);
30   // Also add an "indexing operator".  See discussion in the implementation
31   // also note the return type.  Will discuss in class
32   int& operator[](unsigned);
33
34 private:
35   // Note the use of "private:" above.  This is discussed in the
36   // comments in the main program.
37
38   // Define the variables needed for the queue
39   int      queue[MAX_QUEUE_SIZE];  // The actual data in the queue
40   unsigned length;                 // Number of elements in the queue
41 };
42
43 // Implement the member functions for the LIFOQueue object
44 // Constructor
45 LIFOQueue::LIFOQueue()
46   : length(0)             // Set length to zero
47 { // We can leave the "queue" variable uninitialized.  WHy?
48 }
49
50 bool LIFOQueue::Enque(int newElement)
51 {
52   // First check if queue is full.
53   if (length == MAX_QUEUE_SIZE) return false; // Cannot enque
54   // Ok to enque this new value
55   queue[length++] = newElement;
56   // The line below (commented out) is equivalent
```

Program member-functions.cc

```
57    //this->queue[this->length++] = newElement;
58
59    // Note that when we refer to a variable in a member function,
60    // the compiler automatically assumes we mean the member variable
61    // if there is one by that name.  Also notice that all member functions
62    // always define a pointer "this", that points to an object of the
63    // correct type (LIFOQueue in this case).
64    return true;
65  }
66
67  int LIFOQueue::Deque()
68  {
69    // First check if queue is empty.
70    if (length == 0)
71      { // It's hard to decide what to do here.  "This should never happen".
72        cout << "OOps, Deque called on empty queue" << endl;
73        // For this example we will just exit the program.
74        exit(0);
75      }
76    return queue[--length];
77    // As above, we are refering to member variables by name. eg. "queue"
78    // and "length".  Also as before we could have said:
79    //return this->queue[--(this->length)];
80  }
81
82  unsigned LIFOQueue::Length() const
83  { // Return the length (number of elements) in the queue
84    return length;
85  }
86
87  bool LIFOQueue::Empty() const
88  { // Return true if empty (length is zero)
89    return length == 0;
90  }
91
92  void LIFOQueue::Clear()
93  { // Remove all elements in the queue
94    // Here we would be tempted to write:
95    // while (!Empty()) { Deque();}
96    // Does the below do exactly the same thing?
97    // Which is best?
98    length = 0;
99  }
100
101 void LIFOQueue::Print() const
102 {
103   // Notice that in a member function, we can refer to other member
104   // functions in the same class by name.  eg. "Empty()" below.
105   if (Empty())
106     {
107       cout << "Queue is empty" << endl;
108       return;
109     }
110   for (int i = 0; i < Length(); ++i)
111     {
112       cout << "Element " << i << " is " << queue[i] << endl;
```

Program member-functions.cc (continued)

```
113       }
114   }
115
116   // Implement the addition operator
117   LIFOQueue LIFOQueue::operator+(const LIFOQueue& rhs)
118   { // Notice the left hand side argument is "this"
119     // Make a new LIFOQueue and add all elements in lhs + rhs
120     LIFOQueue returnValue;
121     // Add the left-hand-side values (from "this")
122     for (int i = 0; i < Length(); ++i)
123       { // Add all elements from "this"
124         returnValue.Enque(queue[i]);
125       }
126     // Add the right-hand-side values
127     for (int i = 0; i < rhs.Length(); ++i)
128       { // Add all elements from "rhs"
129         returnValue.Enque(rhs.queue[i]);
130       }
131     return returnValue;
132   }
133
134   // Implement the indexing operator
135   int& LIFOQueue::operator[](unsigned index)
136   { // return the value at the "index" element
137     // The length check below, uncomment if desired
138     //if (index >= length) exit(); // !! Not clear what to do here
139     return queue[index];
140   }
141
142   int main()
143   { // Illustrate the use of the member functions
144     LIFOQueue q1;
145     // Add elements until the queue fills up
146     int i = 0;
147     while(true)
148       {
149         if (!q1.Enque(i++)) break;  // If q1.Enque() is false, queue is full
150       }
151     // Note the syntax above and below.  variable name DOT function name
152     // It calls the "Print()" function with "this" as "address of q1"
153     q1.Print();
154     // Remove elements one at a time
155     while (!q1.Empty())
156       {
157         int k = q1.Deque();
158         cout << "Dequeued element " << k << endl;
159       }
160
161     // Add element 2 ten times
162     for (int i = 0; i < 10; ++i)
163       {
164         q1.Enque(2);
165       }
166     // Define a second queue
167     LIFOQueue q2;
168     // Add element 3 ten times to q2
```

Program member-functions.cc (continued)

```
169     for (int i = 0; i < 10; ++i)
170       {
171         q2.Enque(3);
172       }
173     // Use the addition operator
174     LIFOQueue q3 = q1 + q2;
175     // The above called the addition operator with "address of q1" as the
176     // "this", and q2 as the "right-hand-side"
177     cout << "Printing q3 after addition" << endl;
178     q3.Print();
179     // Add again in reverse order
180     LIFOQueue q4 = q2 + q1;
181     cout << "Printing q4 after addition" << endl;
182     q4.Print();
183     // So, what about the addition operator?  Did our implemenation make sense?
184     // What might have been a better choice?
185     // Call the copy constructor
186     LIFOQueue q5(q4);
187     // We did not define a copy constructor, so what happened above?
188     cout << "Printing q5 after copy constructor" << endl;
189     q5.Print();
190     // Illustrate the indexing operator
191     LIFOQueue q6;
192     // First populate the queue with some values
193     for (int i = 0; i < 10; ++i)
194       {
195         q6.Enque(i);
196       }
197     // Now use indexing operator to retrieve them
198     for (unsigned i = 0; i < q6.Length(); ++i)
199       {
200         int v = q6[i];  // Return the "i'th" element of q6
201         cout << "Element " << i << " is " << v << endl;
202       }
203
204     // Now use indexing operator on LEFT HAND SIDE!
205     for (unsigned i = 0; i < q6.Length(); ++i)
206       {
207         q6[i] = q6[i] * 10;
208       }
209     // And print it out
210     cout << "After LHS indexing loop" << endl;
211     for (unsigned i = 0; i < q6.Length(); ++i)
212       {
213         int v = q6[i];  // Return the "i'th" element of q6
214         cout << "Element " << i << " is " << v << endl;
215       }
216
217
218
219     // The fact that member variables "length" and "queue" are declared
220     // "private" means that they cannot be referenced except in member
221     // functions. In other words, the below (commented out) would not
222     // compile if it was uncommented.
223     // int q1Length = q1.length;
224     // Even though q1.length definitely exists, since it is private it
```

Program member-functions.cc (continued)

```
225     // cannot be referenced here.  However, we defined a member functiion
226     // "Length() const" that returned the value of length. Why is this
227     // better?
228     }
```

Program member-functions.cc (continued)