Data Structures and Algorithms

Chapter 4

Queues

Lecture Contents

- Introduction to Queues
- Designing and Building a Queue Class
  - Array Based
- Linked Queues
- Priority Queues

Lecture Objectives

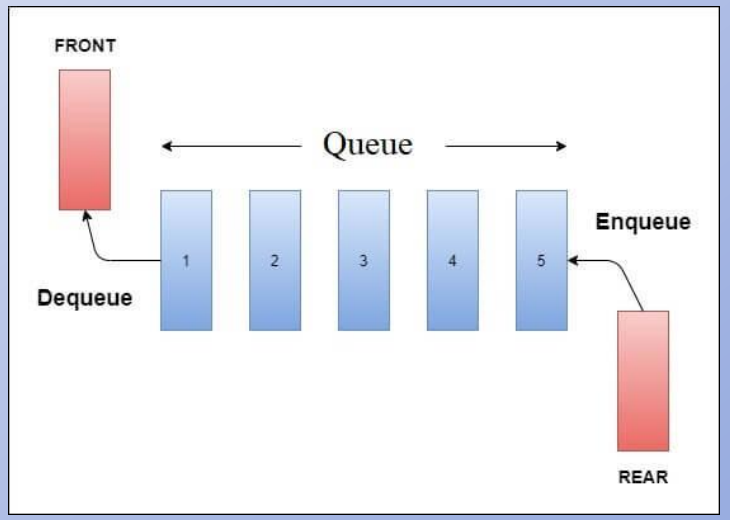
- To study a queue as an ADT
- Build a static-array-based implementation of queues
- Build a dynamic-array-based implementation of queues
- Build a Linked list-based implementation of queues

Introduction to Queues

- A queue is a waiting line – seen in daily life
  - A line of people waiting for a bank teller
  - A line of cars at a toll booth
  - "This is the captain, we're 5th in line for takeoff"
- What other kinds of queues can you think of

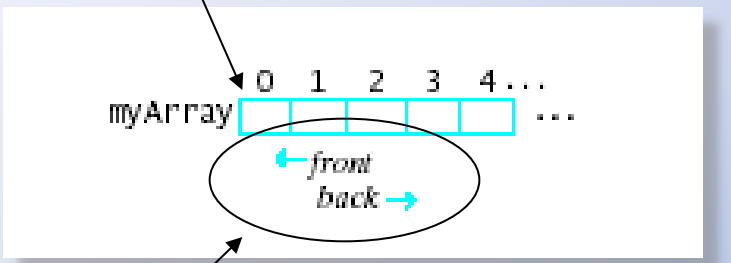
The Queue As an ADT

- A queue is a sequence of data elements
- In the sequence
  - Items can be removed only at the front
  - Items can be added only at the other end, the back
  - A queue exhibits *First-In-First-Out (FIFO)*
- Basic operations
  - Construct a queue
  - Check if empty
  - Enqueue (add element to back)
  - Front (retrieve value of element from front)
  - Dequeue (remove element from front)

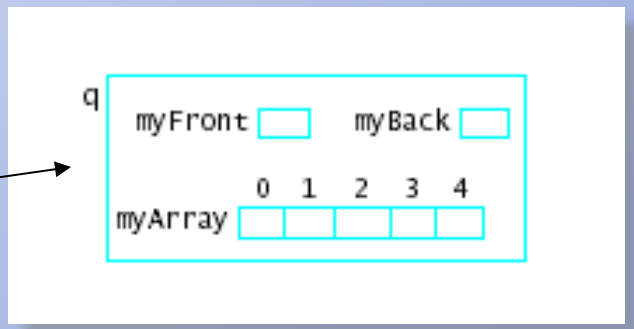


Designing and Building a Queue Class Array-Based

- Consider an array in which to store a queue

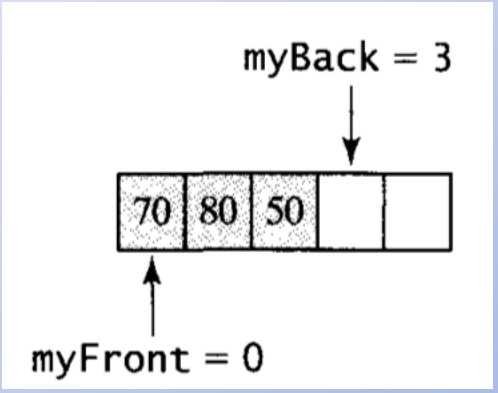


- Note additional variables needed
  - **myFront, myBack**
- Picture a queue object like this

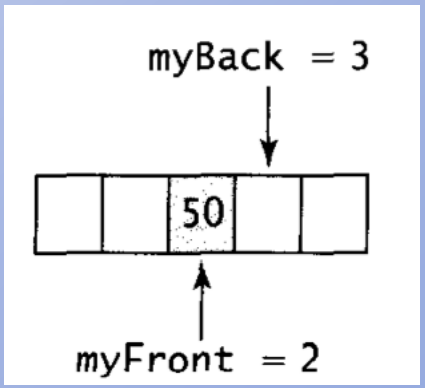


Designing and Building a Queue Class Array-Based

- The sequence of operations add 70, add 80, add 50 produces the following configuration:

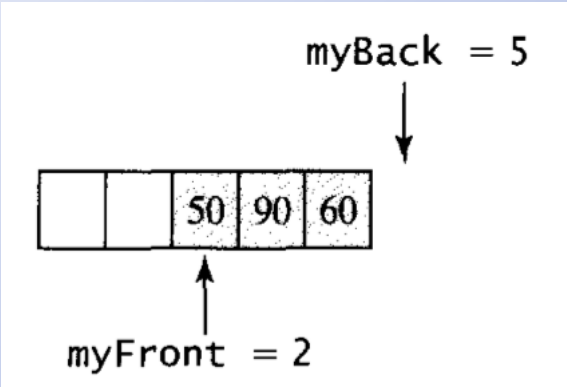


- Now suppose that two elements are removed

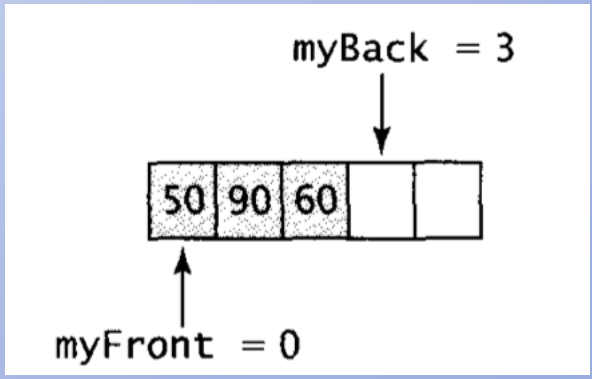


Designing and Building a Queue Class Array-Based

- and that 90 and 60 are then added



- Before another item can be inserted into the queue, the elements in the array must be shifted back to the beginning of the array



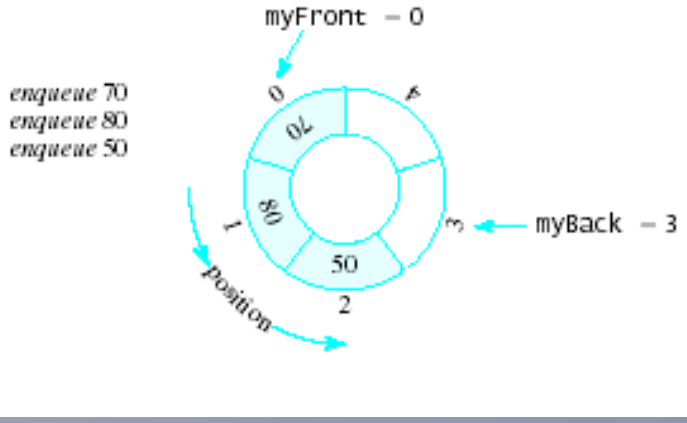
Designing and Building a Queue Class Array-Based

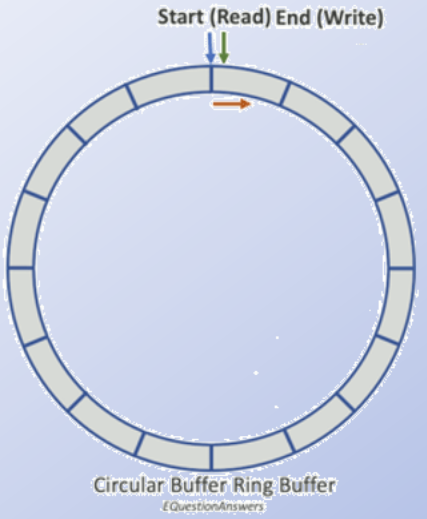
- Problems
  - We quickly "walk off the end" of the array

- Possible solutions
  - Shift array elements
  - Note that both empty and full queue gives `myBack == myFront`

Designing and Building a Queue Class Array-Based

- Problems
  - We quickly "walk off the end" of the array
- Possible solutions
  - Use a circular queue



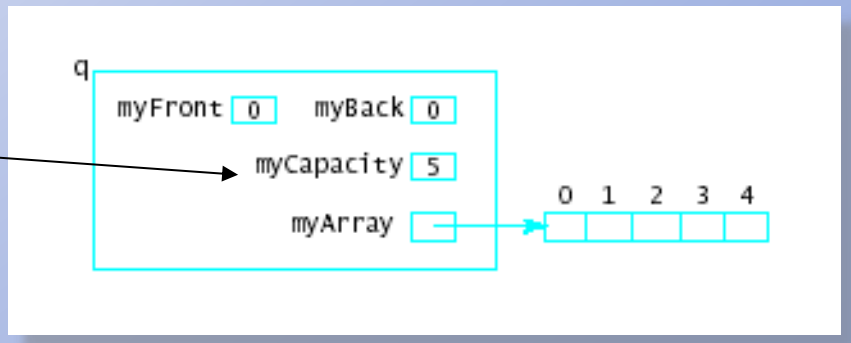


Designing and Building a Queue Class Array-Based

- Using a static array
  - **`QUEUE_CAPACITY`** specified
  - Enqueue increments **`myBack`** using mod operator, checks for full queue
  - Dequeue increments **`myFront`** using mod operator, checks for empty queue
- Note declaration of Queue class,
- View implementation,

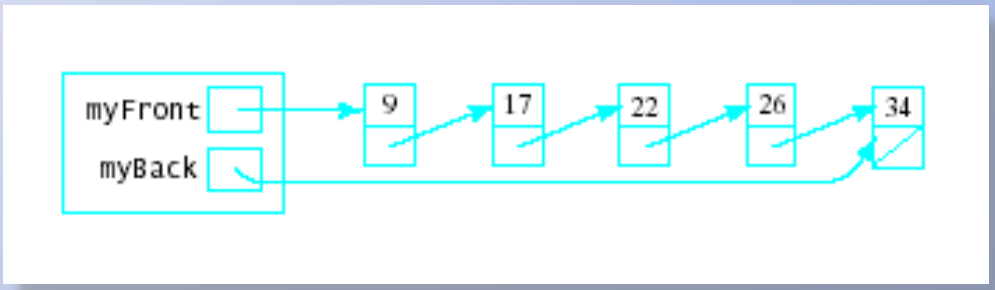
Using Dynamic Array to Store Queue Elements

- Similar problems as with list and stack
  - Fixed size array can be specified too large or too small
- Dynamic array design allows sizing of array for multiple situations
- Results in structure as shown
  - **`myCapacity`** determined at run time



Linked Queues

- Even with dynamic allocation of queue size
  - Array size is still fixed
  - Cannot be adjusted during run of program
- Could use linked list to store queue elements
  - Can grow and shrink to fit the situation
  - No need for upper bound (`myCapacity`)

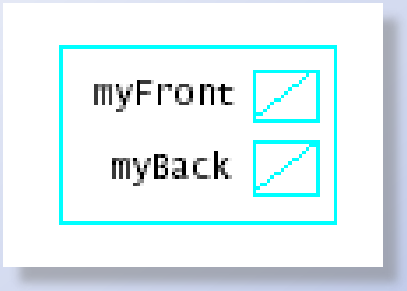


Linked Queues

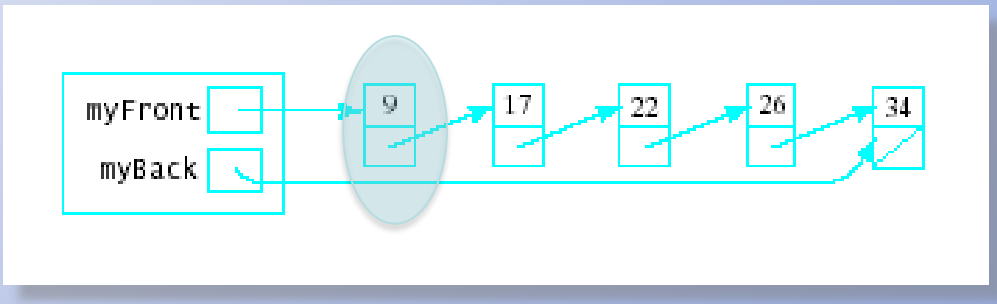
- Constructor

  initializes

  `myFront, myBack=null`

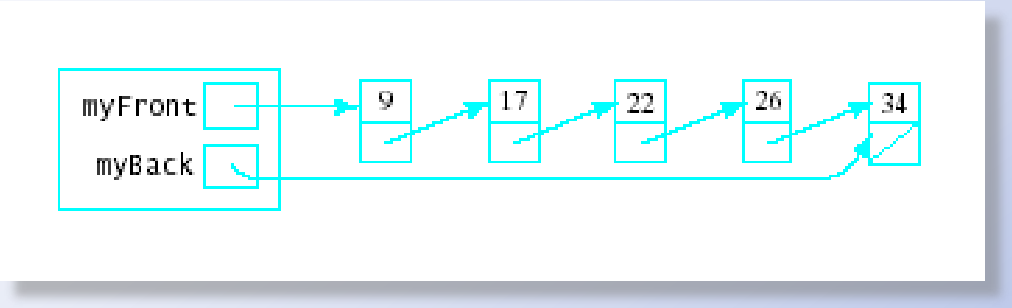


- Front



  - **`return myFront->data`**

Linked Queues

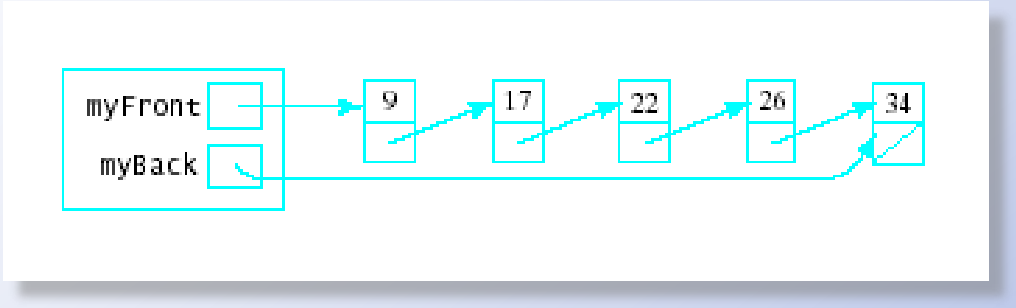


➢ Enqueue

- ❖ Insert node at end of list (Watch for the first element)

```
87  //--- Definition of enqueue()
88  void Queue::enqueue(const QueueElement & value)
89  {
90      Queue::NodePointer newptr = new Queue::Node(value);
91      if (empty())
92          myFront = myBack = newptr;
93      else
94      {
95          myBack->next = newptr;
96          myBack = newptr;
97      }
98  }
99
```

Linked Queues



➢Dequeue

❖Delete first node (watch for empty queue)

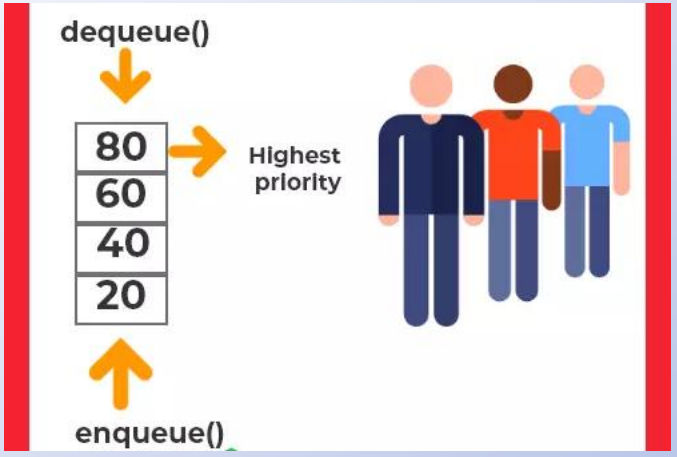
```
132  //--- Definition of dequeue()
133  void Queue::dequeue()
134  {
135      if (!empty())
136      {
137          Queue::NodePointer ptr = myFront;
138          myFront = myFront->next;
139          delete ptr;
140          if (myFront == 0)      // queue is now empty
141              myBack = 0;
142      }
143      else
144          cerr << "*** Queue is empty -- can't remove a value ***\n";
145  }
146
```

Priority Queues

- Priority Queue is more specialized data structure than Queue.
- Priority queue has same method but with a major difference.
- Items are ordered by *key* or value so that item with the *lowest key or value* is at front and item with the *highest key or value* is at back or vice versa.
- Example: CIB Segments (Prime, Plus, Wealth, Private).

Priority Queues

- Priority Queue with Value:



- Priority Queue with Key: