Data Structure and Algorithms

Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

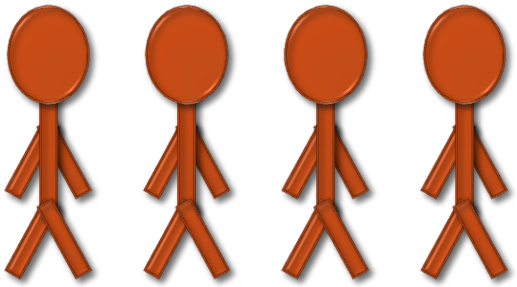
Queues

- Queue is First-In-First-Out (FIFO) data structure
  - First element added to the queue will be first one to be removed
- Queue implements a special kind of list
  - Items are inserted at one end (the rear)
  - Items are deleted at the other end (the front)

Queue – Analogy

- A queue is like a line of people waiting for a bank teller
- The queue has a front and a rear

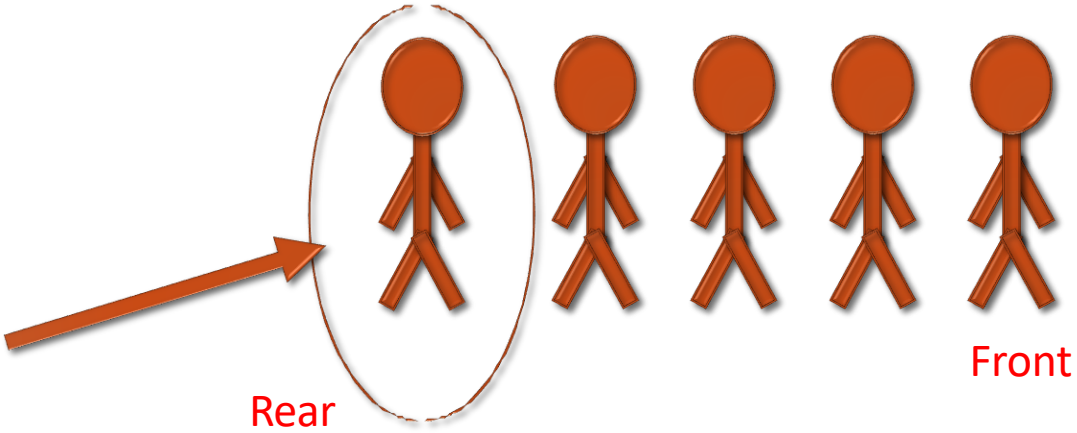
Rear



Front

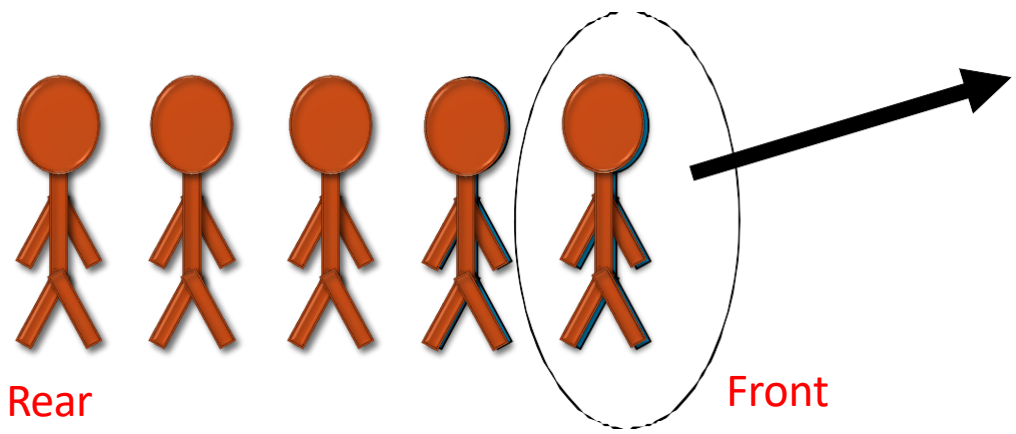
Queue – Analogy

- New people must enter the queue at the rear



Queue – Analogy

- An item is always taken from the front of the queue



Queues – Examples

- Billing counter
  - Booking movie tickets
  - Queue for paying bills
- A print queue
- Vehicles on toll-tax bridge
- Luggage checking machine
- And others?

Queues – Applications

- Operating systems
  - Process scheduling in multiprogramming environment
  - Controlling provisioning of resources to multiple users (or processing)

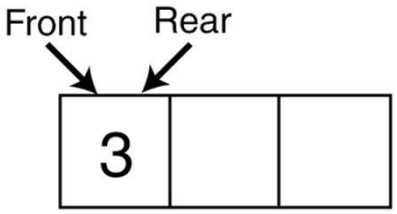
- Middleware/Communication software
  - Hold messages/packets in order of their arrival
    - ➢ Messages are usually transmitted faster than the time to process them
  - The most common application is in client-server models
    - ➢ Multiple clients may be requesting services from one or more servers
    - ➢ Some clients may have to wait while the servers are busy
    - ➢ Those clients are placed in a queue and serviced in the order of arrival

Basic Operations (Queue ADT)

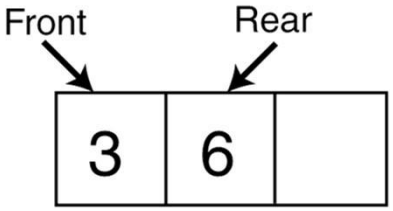
- MAKENULL(Q)
  - Makes Queue Q be an empty list
- FRONT(Q)
  - Returns the first element on Queue Q
- ENQUEUE(x,Q)
  - Inserts element x at the end of Queue Q
- DEQUEUE(Q)
  - Deletes the first element of Q
- EMPTY(Q)
  - Returns true if and only if Q is an empty queue

Enqueue And Dequeue Operations

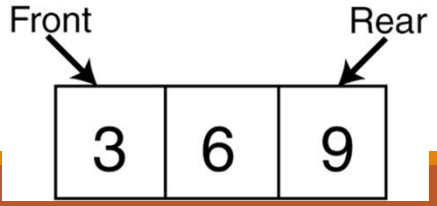
Enqueue(3);



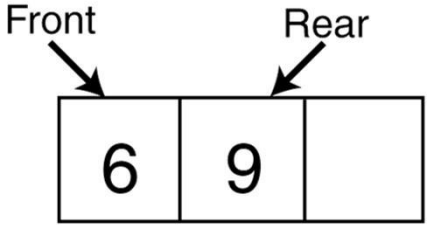
Enqueue(6);



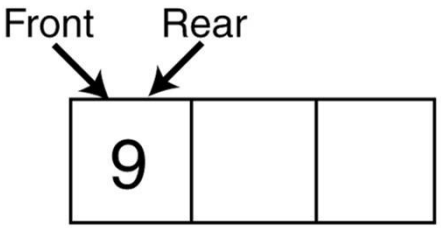
Enqueue(9);



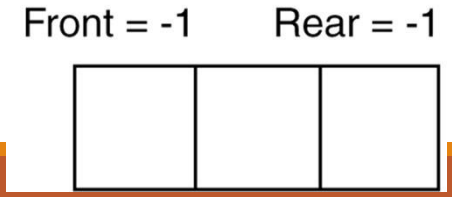
Dequeue();



Dequeue();



Dequeue();



Implementation

- Static
  - Queue is implemented by an array
  - Size of queue remains fix

- Dynamic
  - A queue can be implemented as a linked list
  - Expand or shrink with each enqueue or dequeue operation

Array Implementation

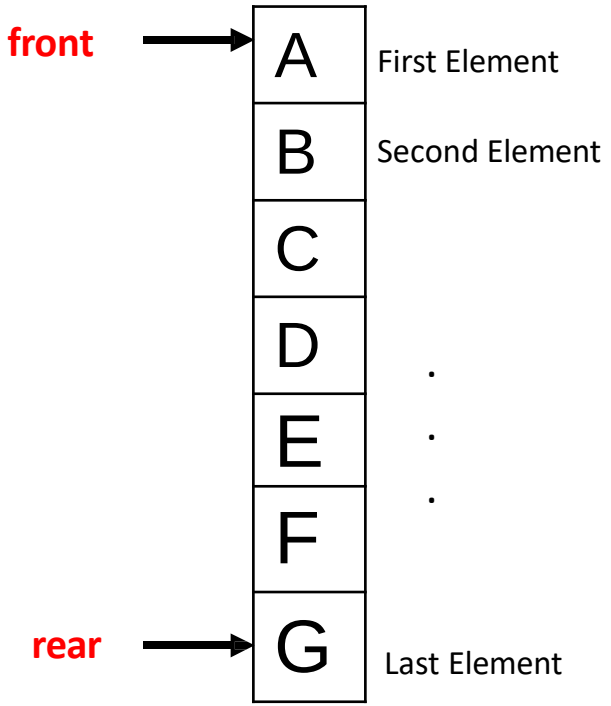
- Use two counters that signify rear and front

- When queue is empty
  - Both front and rear are set to -1

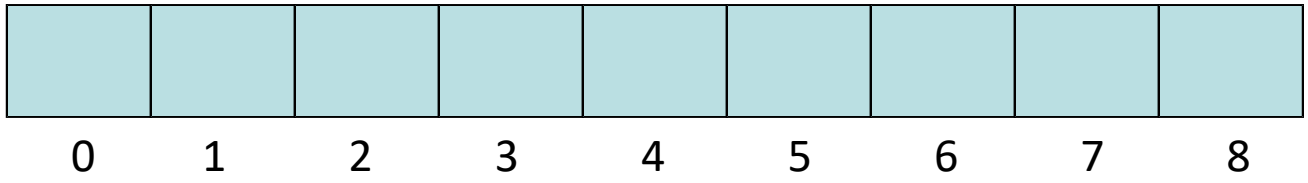
- When there is only one value in the Queue,
  - Both rear and front have same index

- While enqueueing increment rear by 1

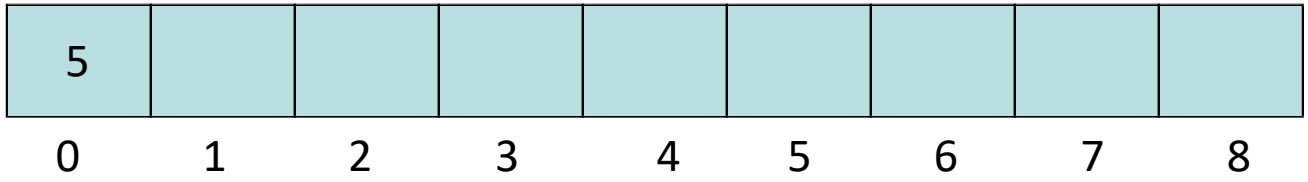
- While dequeueing, increment front by 1



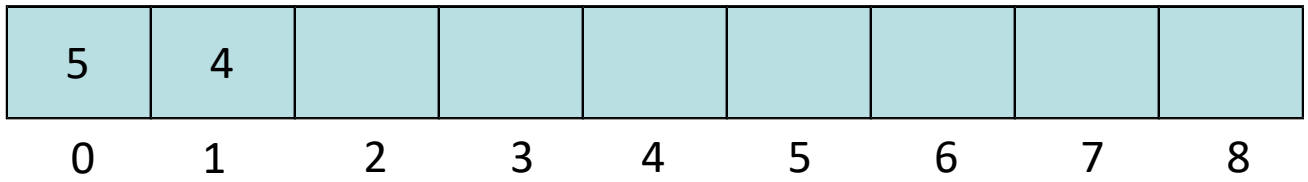
Array Implementation Example



front= -1
rear = -1

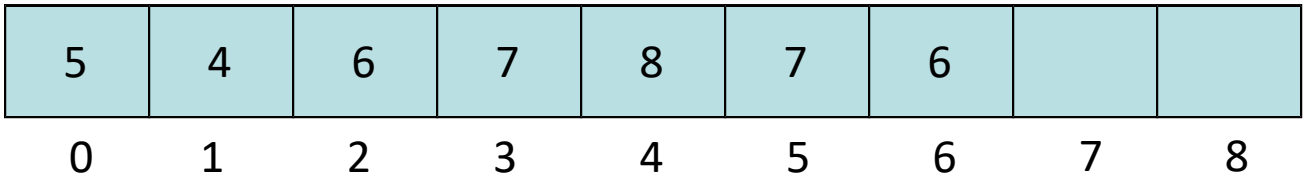


front= 0
rear = 0

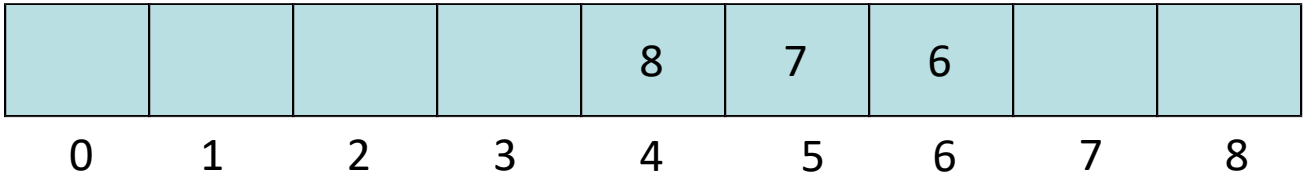


front= 0
rear = 1

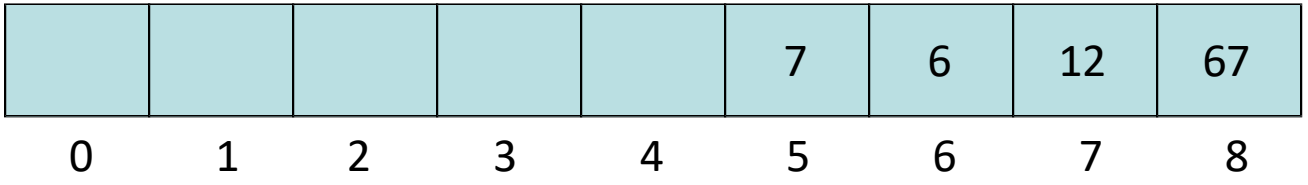
Array Implementation Example



front=0
rear=6



front=4
rear=6



front=5
rear=8

Problem: How can we insert more elements?
Rear index can not move beyond the last element….

Using Circular Queue

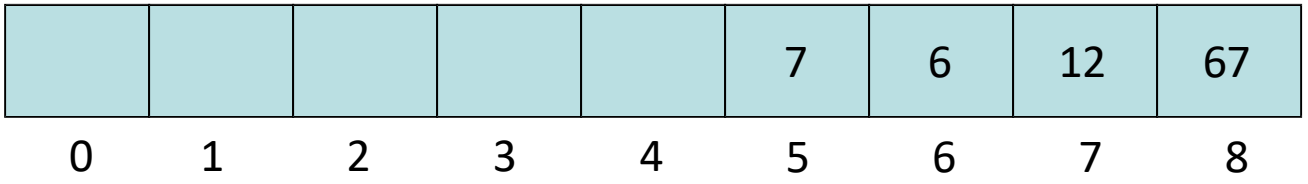
- Allow rear to wrap around the array

```
if(rear == queueSize-1)
        rear = 0;
else
        rear++;
```

- Alternatively, use modular arithmetic

```
rear = (rear + 1) % queueSize;
```

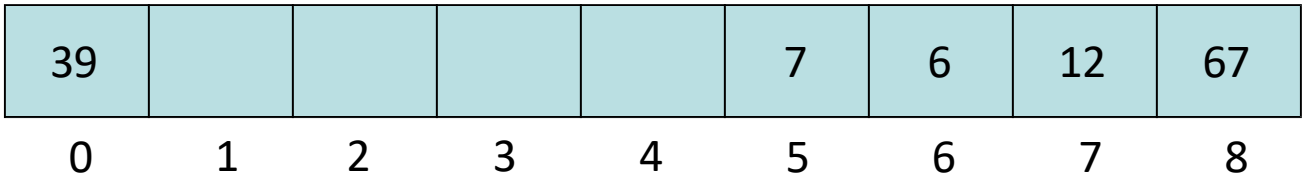
Example



front=5
rear=8

Enqueue 39

- Rear = (Rear+1) mod queueSize = (8+1) mod 9 = 0



front=5
rear=0

Problem: How to avoid overwriting an existing element?

Array Implementation – Code

```
class IntQueue
{
    private:
    int *queueArray;
    int queueSize;
    int front;
    int rear;
    int numItems;
    public:
        IntQueue(int);
        ~IntQueue(void);
        void enqueue(int);
        int dequeue(void);
        bool isEmpty(void);
        bool isFull(void);
        void makeNull(void); };
```

Array Implementation – Code

```
class IntQueue
{
    private:
    int *queueArray;
    int queueSize;
    int front;
    int rear;
    int numItems;
    public:
      IntQueue(int);
      ~IntQueue(void);
      void enqueue(int);
      int dequeue(void);
      bool isEmpty(void);
      bool isFull(void);
      void makeNull(void); };
```

Clears the queue by resetting the frontand rear indices, and setting the numItems to 0.

Array Implementation – Code

Constructor

```
IntQueue::IntQueue(int s) {
    queueArray = new int[s];
    queueSize = s;
    front = -1;
    rear = -1;
    numItems = 0;
}
```

Destructor

```
IntQueue::~IntQueue(void)
{
    delete [] queueArray;
}
```

Array Implementation – Code

- isFull() returns true if the queue is full and false otherwise

```
bool IntQueue::isFull(void)
{
        if (numItems < queueSize)
                return false;
        else
                return true;
}
```

- makeNull() resets front & rear indices and sets numItems= 0

```
void IntQueue::makeNull(void)
{
        front = - 1; rear = - 1; numItems = 0;
}
```

Array Implementation – Code

- Function enqueue inserts the value in num at the end of the Queue

```
void IntQueue::enqueue(int num)
{
    if (isFull())
        cout << "The queue is full.\n";

    else {
        // Calculate the new rear position
        rear = (rear + 1) % queueSize;
        // Insert new item
        queueArray[rear] = num;
        // Update item count
        numItems++;
    }
}
```

Array Implementation – Code

- Function dequeue removes and returns the value at the front of the Queue

```
int IntQueue::dequeue(void)
{
int num;
if (isEmpty())
cout << "The queue is empty.\n";
else {
// Move front
front = (front + 1) % queueSize;
// Retrieve the front item
num = queueArray[front];
// Update item count
numItems--;
}
return num;
}
```

Using Queues

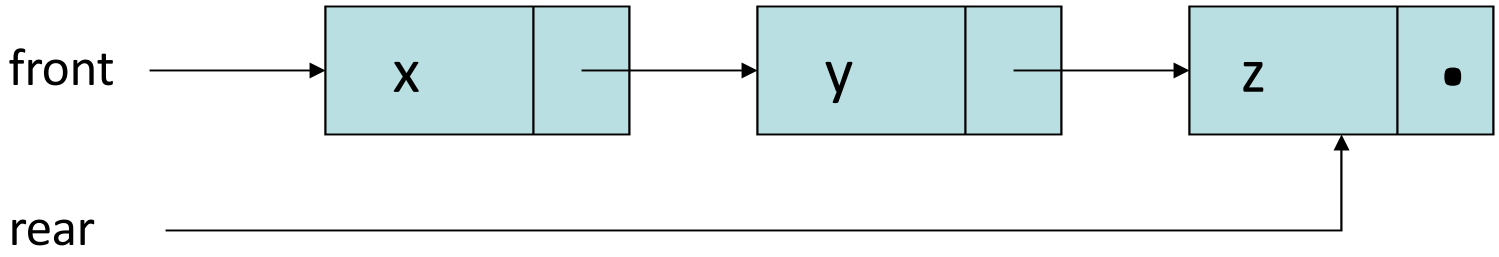
```
void main(void)
{
    IntQueue iQueue(5);
    cout << "Enqueuing 5 items...\n";
    // Enqueue 5 items.
    for (int x = 0; x < 5; x++)
    iQueue.enqueue(x);
    // Attempt to enqueue a 6th item.
    cout << "Now attempting to enqueue again...\n";
    iQueue.enqueue(5);
    // Deqeue and retrieve all items in the queue
    cout << "The values in the queue were:\n";
    while (!iQueue.isEmpty()){
        int value;
        value = iQueue.dequeue();
        cout << value << endl;
    }
}
```

Output:

```
Enqueuing 5 items...
Now attempting to enqueue again...
The queue is full
The values in the queue were: 0
1
2
3
4
```

Pointer-Based Implementation

- Queue Class maintains two pointers
  - front: A pointer to the first element of the queue
  - rear: A pointer to the last element of the queue



Queue Operations

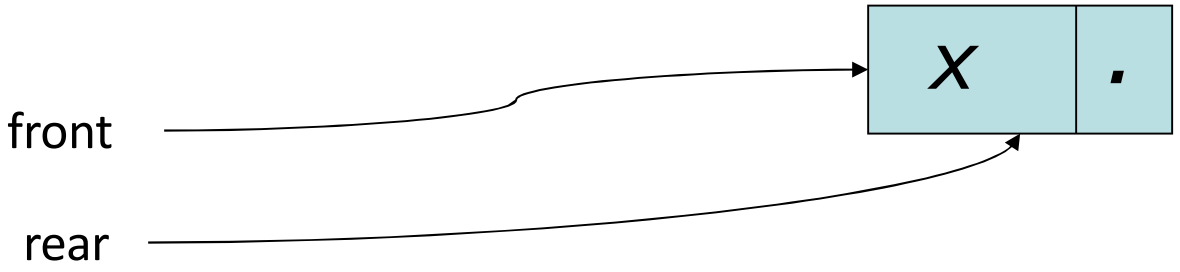
- MAKENULL(Q)

NULL

front

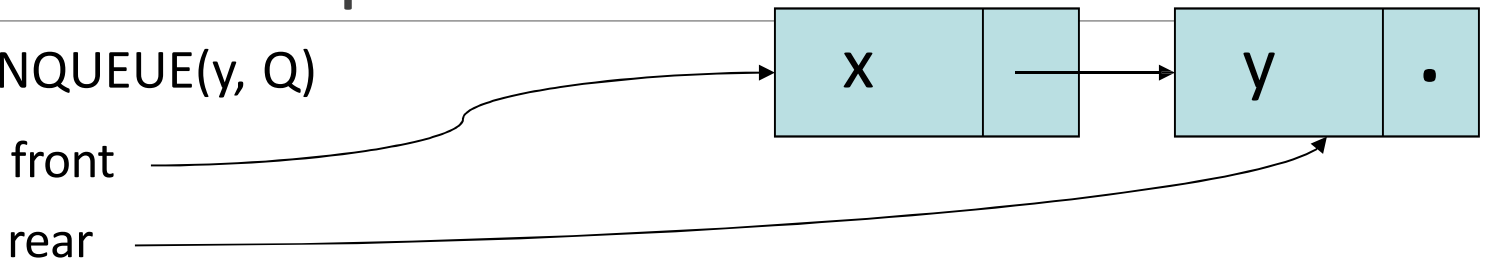
rear

- ENQUEUE (x, Q)

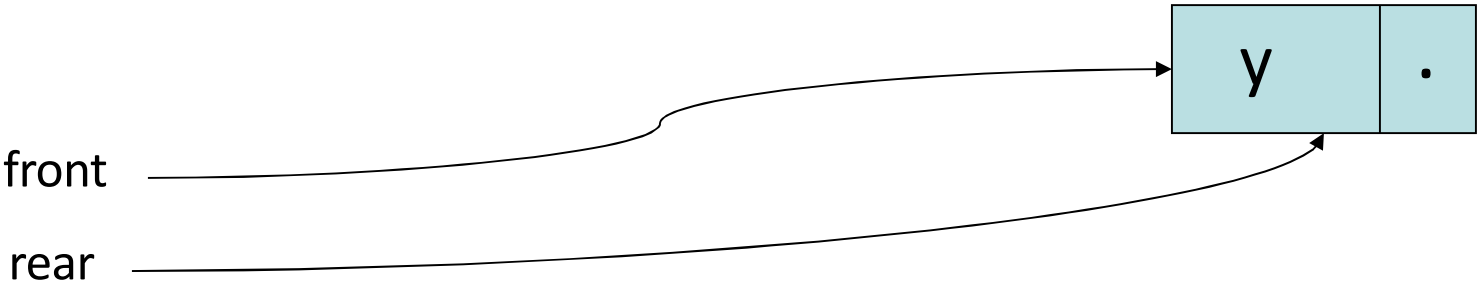


Queue Operations

- ENQUEUE(y, Q)



- DEQUEUE (Q)



Pointer Implementation – Code

```
class DynIntQueue
{
private:
      struct QueueNode
      {
            int value;
            QueueNode *next; };
      QueueNode *front;
      QueueNode *rear;
      int numItems; public:
      DynIntQueue(void);
      ~DynIntQueue(void);
      void enqueue(int);
      int dequeue(void);
      bool isEmpty(void);
      void makeNull(void);
};
```

Pointer Implementation – Code

- Constructor

```
DynIntQueue::DynIntQueue(void)
{
      front = NULL;
      rear = NULL;
      numItems = 0;
}
```

- isEmpty() returns true if the queue is full and false otherwise

```
bool DynIntQueue::isEmpty(void)
{
      if (numItems)
            return true;
      else
            return false;
}
```

Pointer Implementation – Code

- Function enqueue inserts the value in num at the end of Queue

```
void DynIntQueue::enqueue(int num)
{
    QueueNode *newNode;
    newNode = new QueueNode;
    newNode->value = num;
    newNode->next = NULL;
    if (isEmpty()) {
        front = newNode;
        rear = newNode;
    }
    else {
        rear->next = newNode;
        rear = newNode;
    }
    numItems++; }
```

Pointer Implementation – Code

- Function dequeue removes and returns the value at the front of the Queue

```
int DynIntQueue::dequeue(void)
{
    QueueNode *temp;
    int num;
    if (isEmpty())
        cout << "The queue is empty.\n";
    else {
        num = front->value;
        temp = front->next;
        delete front;
        front = temp;
        numItems--;
    }
    return num;
}
```

Pointer Implementation – Code

- Destructor

```
DynIntQueue::~DynIntQueue(void)
{
    makeNull();
}
```

- makeNull() resets front& rearindices to NULLand sets numItemsto 0

```
void DynIntQueue::makeNull(void)
{
    while(!isEmpty()){
        dequeue();
    }
}
```

Using Queues

```
void main(void)
{
DynIntQueue iQueue;
cout << "Enqueuing 5 items...\n";
// Enqueue 5 items
for (int x = 0; x < 5; x++)
iQueue.enqueue(x);
// Deqeue and retrieve all items in the queue
cout << "The values in the queue were:\n";
while (!iQueue.isEmpty())
    {
        int value;
        value= iQueue.dequeue();
        cout << value << endl;  } }
```

Output:

```
Enqueuing 5 items...
The values in the queue were: 0
1
2
3
4
```

Any Question So Far?