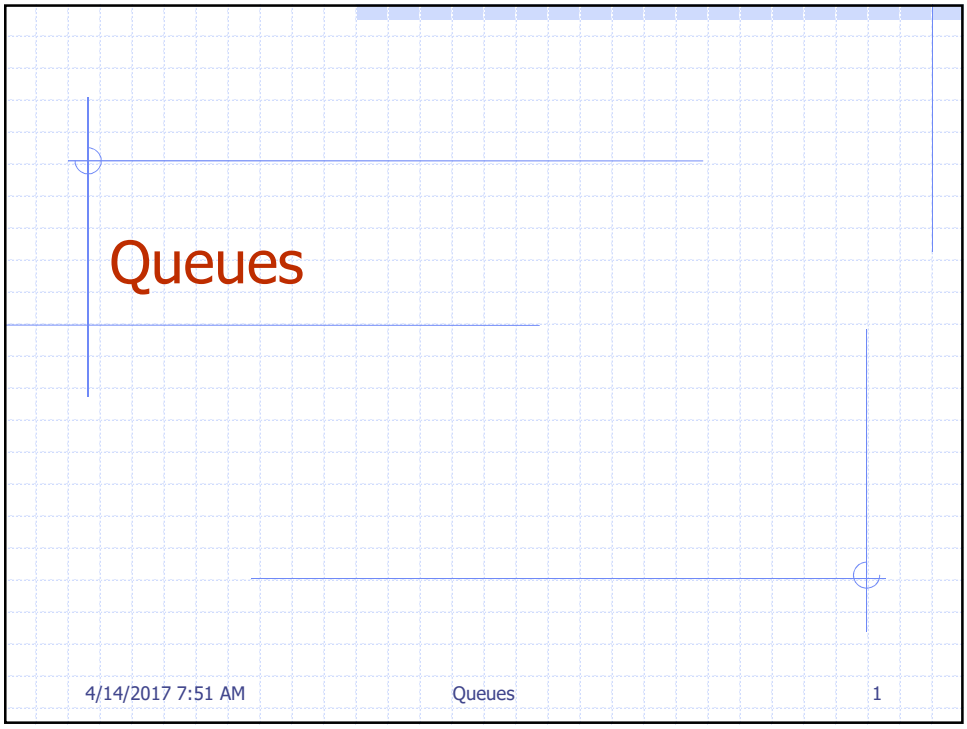


Queue ADT

- ***Definition:*** a **queue** is a collection of objects that are inserted and removed according to the first-in-first-out (FIFO) principle.
- Objects are inserted into the **rear** of the queue.
- Objects can **ONLY** be removed from the **front** of the queue.
- Objects that have been in the queue the longest are first to be removed.

The Queue ADT

- Examples of queues
  - Lines
    - Ticket line, amusement park line, etc
  - Access to shared resources (e.g., printer queue)
  - Phone calls to large companies
  - Waiting list for adding classes

The Queue ADT

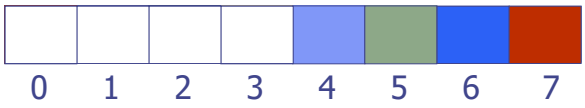
- Main queue operations:
  - enqueue: inserts an object at the end of the queue
  - dequeue: just removes the element at the front of the queue
  - getFront: returns the element at the front without removing it
- Auxiliary queue operations:
  - size: returns the number of elements stored - either keep a counter or calculate
  - isEmpty: returns a Boolean indicating whether no elements are stored

Naïve Array-based Queue

- Two variables keep track of the front and rear
  - *front* index of the front element, initialize to 0
  - *rear* index immediately past the rear element, initialize to 0
- Counter to keep track of number of items in the queue.

Naïve Array-based Queue

front = 4
rear = 8



What happens on the next enqueue operation?
What are the possible solutions?

Circular Array-based Queue

- Best solution - use a wrapping or circular array
  - Enqueue at the beginning of the array

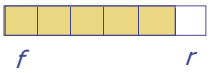
front = 4
rear = 3

Queue Data Structure

```
class Queue
{
private:
        objectType queue[MAX_QUEUE_SIZE];
        int front;
        int rear;
        int size;

public:
        functions for queue manipulation
        constructor sets front and rear to 0
};
```

Queue Operations - Enqueue




```
void enqueue ( const Object &o )
        if ( front == ( rear + 1 ) % size )
                error – queue full
        else
                Q[rear] = o
                rear = ( rear + 1 ) % size
```

0 = (5 + 1) % 6

2 = (1 + 1) % 6

Queue Operations - Dequeue

- In class exercise - Write the code for dequeue and getFront.
  - const getFront will return front object
    - *const Object & getFront ( ) const*
  - dequeue will just remove the front object
    - *void dequeue ( )*

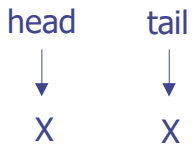
Growable Array-based Queue

- In an enqueue operation, when the array is full, instead of making this an error condition, we can replace the array with a larger one
- The enqueue operation has amortized running time
  - $O(n)$ with the incremental strategy
  - $O(1)$ with the doubling strategy

Linked List Based Queue

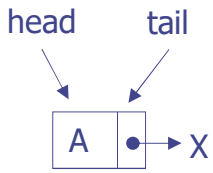
- Using a linked list can remove the size restrictions of an array
- Linked list with front and rear pointers
  - front is the same as head, rear is the same as tail
- Front and rear initially point to null

Linked List Based Queue



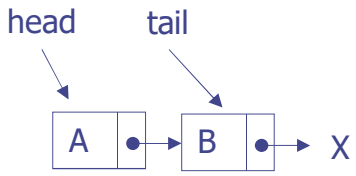
Linked List Based Queue

Enqueue



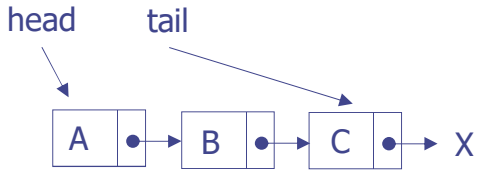
Linked List Based Queue

Enqueue



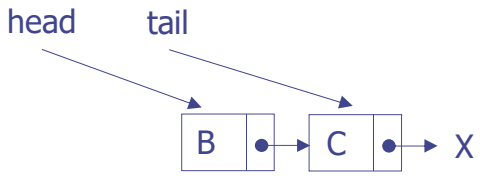
Linked List Based Queue

Enqueue



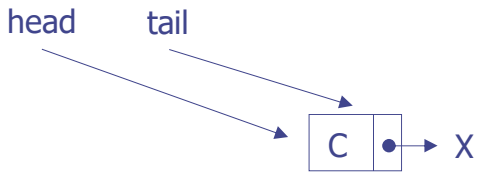
Linked List Based Queue

Dequeue



Linked List Based Queue

Dequeue



Linked List Based Queue

```
class Queue
{
private:
        Node *front;
        Node *rear;
        int numItems;
public:
        all functions to interface with queue
};
```


Linked List Based Queue

```
const Object & getFront( ) const
{
  if( !front )
    error – Queue empty
  else
    return front->data;
}
```

```
bool isEmpty( )
{
  return ( !rear );
}
```

```
void enqueue( const Object &o )
{
  Node *newNode = new Node( o );
  if ( isEmpty( ) )
    front = newNode;
  else
    rear->next = newNode;
  rear = newNode;
}
```

Linked List Based Queue

- In class exercise - write dequeue
  - Just like removing head in a list with head and tail pointers
  - Think about memory leaks
    - Delete the node completely - don't return anything
  - Use getFront( ) if you want to use the object
  - Use dequeue( ) if you just want to remove the node

Linked List Based Queue

Queue Big Oh Runtimes

- Array based
  - Enqueue
  - Dequeue
  - isEmpty
  - getFront
- Linked list based
  - Enqueue
  - Dequeue
  - isEmpty
  - getFront