Queues

Notes from Charlie Obimbo and revised by Yan Yan.

Contents

1. Definition of Queues
2. Queues Implementation

Learning Objectives

1. Define the data structure of the queue ADT
2. Implement the operations of the queue ADT.
3. Describe the advantages and disadvantages of linked list implementation and array implementation of the queue.

Review of Stacks and Queues

- **Stacks & Queues**
  - Stacks: Last in First Out

- Queues: First in First Out

Stack: Definition

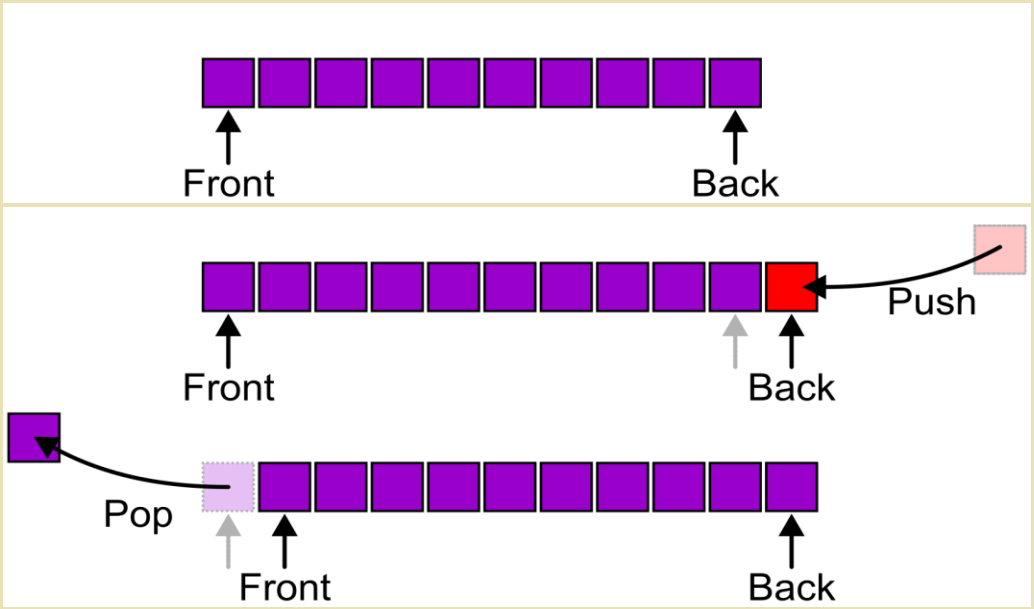
- **Stack**: A stack is an ADT in which items are only inserted on or removed from the top of a stack.
- **Push** operation inserts an item on the top of the stack.
- **Pop** operation removes and returns the item at the top of the stack.
- A *linear* data structure, in which elements are accessed using the LIFO (Last in First Out) Order.

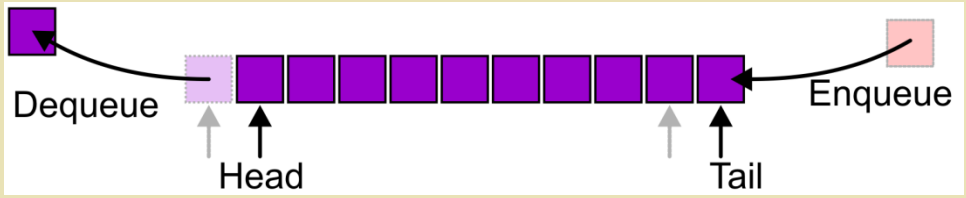
Queue: Definition

- **Queue**: A queue is an ADT in which items are inserted at the end of the queue and removed from the front of the queue.
- **Enqueue** operation inserts an item at the end of the queue.
- **Dequeue** operation removes and returns the item at the front of the queue.
- A *linear* data structure, FIFO (First in First Out).

Queue ADT

- Insertions and removals are performed individually
- The object designated as the front of the queue is the object which was in the queue the *longest*





Figures from course notes by Douglas W. Harder.

Queues: Operations

- **Enqueue(queue, x): inserts x at end of the queue**
- **Dequeue(queue): removes and returns the element at the front of the queue**
- **Peek(queue) (or front): returns the element at the front without removing it**
- **IsEmpty(queue): indicates whether no elements are stored**

Queues: Operations

- Errors:
  - Attempting the execution of dequeue or front on an empty queue, or enqueue on a full queue

Time Complexities

- What are the time complexities of the operations on the queue?

- We require Enqueue(), Dequeue(), isEmpty() and peek() all take O(1) time.

Queues: Exercise

Given numQueue: 5, 9, 1 (front is 5)

- What are the queue contents after the following?

  Enqueue(numQueue, 4)

  Enqueue(numQueue, 7)

- Following the previous operations, What is returned by Dequeue(numQueue)?

What operation determines if the queue contains no items?

Queues: Exercise

Given numQueue: 5, 9, 1 (front is 5)

- What are the queue contents after the following?

  Enqueue(numQueue, 4)

  Enqueue(numQueue, 7) 5,9,1,4,7

- Following the previous operations, What is returned by Dequeue(numQueue)? 5

What operation determines if the queue contains no items? IsEmpty

Queues: Exercise

Given numQueue: 90, 20, 97

- What are the queue's contents after the following operations?
  Enqueue(numQueue, 14)
  Dequeue(numQueue)
  Dequeue(numQueue)

Queues: Exercise

Given numQueue: 90, 20, 97

- What are the queue's contents after the following operations?
  Enqueue(numQueue, 14)
  Dequeue(numQueue)
  Dequeue(numQueue)

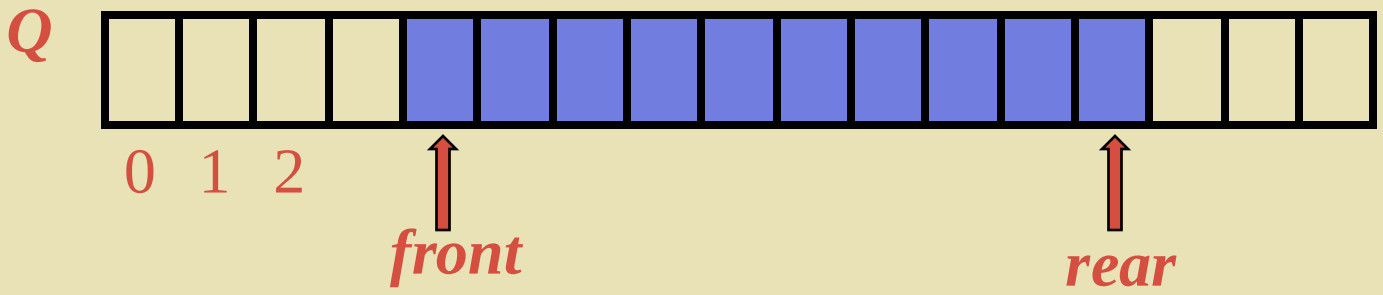
97,14

Queues

- **Implementation:**
  - linked list
  - array

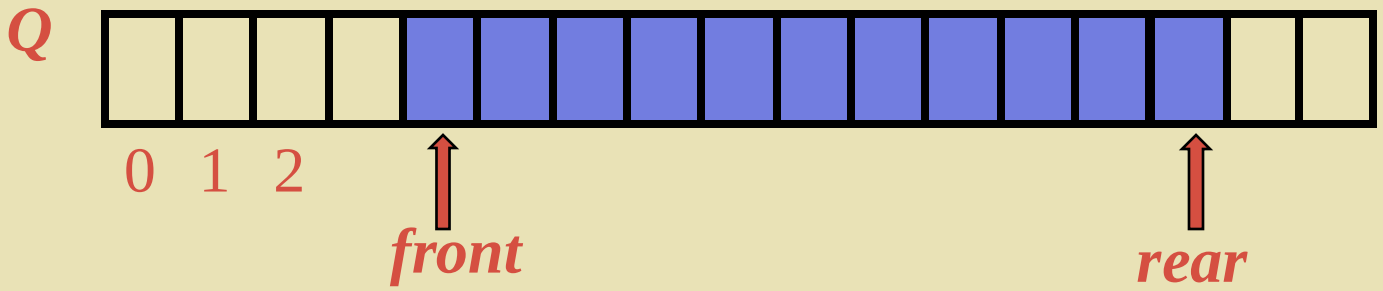
Array-based Implementation

- Define an array
- Q – index between front and rear
  - *rear* – add element
  - *front* – remove element
- Rest of the element in the array are free space



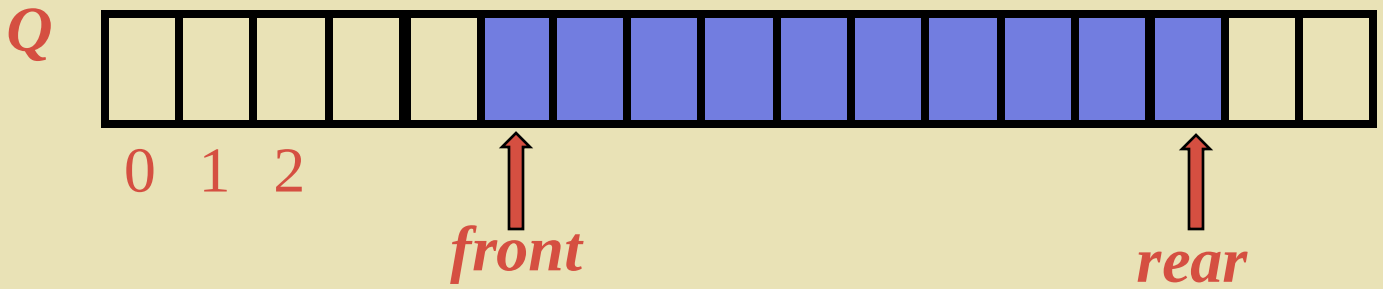
Array-based Implementation

- Define an array
- Q – index between front and rear
  - *rear* – add element
  - *front* – remove element
- Rest of the element in the array are free space
  - Add element to the Q



Array-based Implementation

- Define an array
- Q – index between front and rear
  - *rear* – add element
  - *front* – remove element
- Rest of the element in the array are free space
  - Remove element to the Q

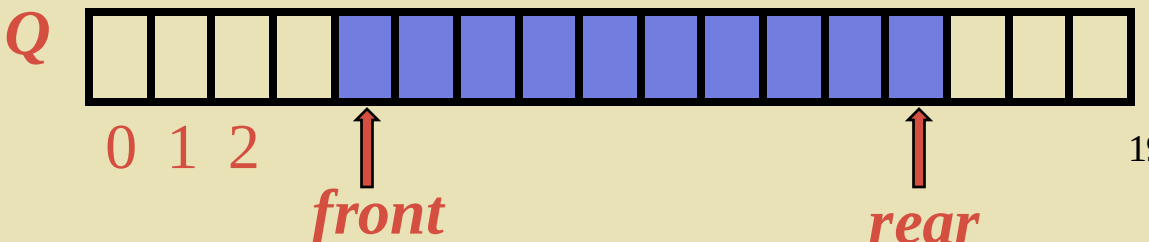


Array-based Implementation

- Pseudocode

```
int A[n]
//initialization
front = -1
rear = -1
```

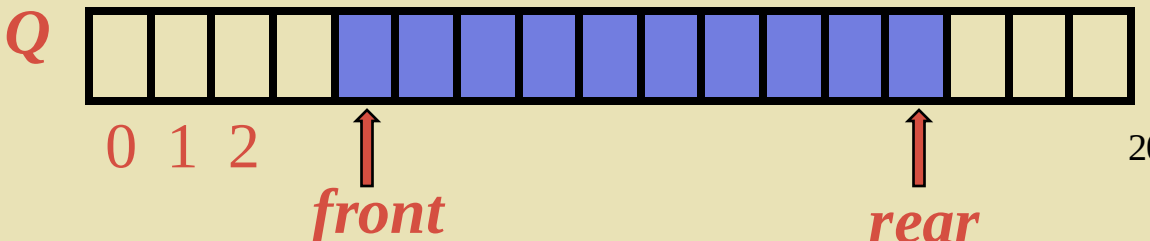
```
//Check if the Queue is empty
IsEmpty () {
if front == -1 && rear == -1
        return true
else
        return false
}
```



Array-based Implementation

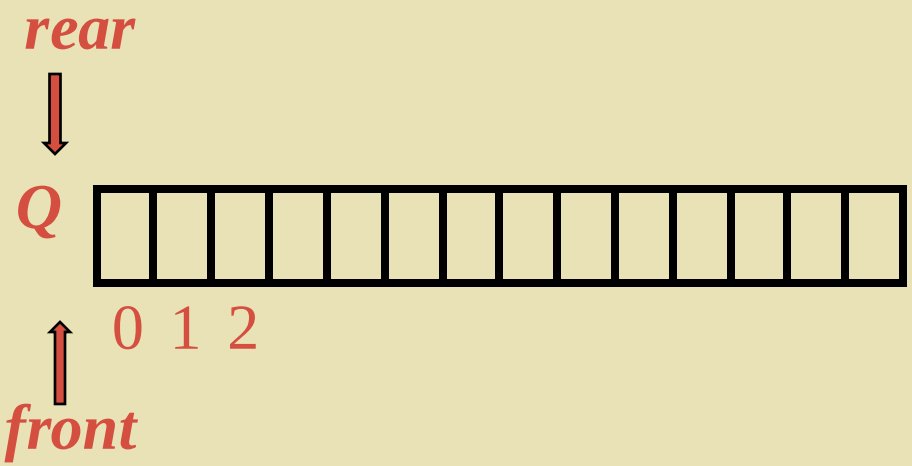
- Pseudocode

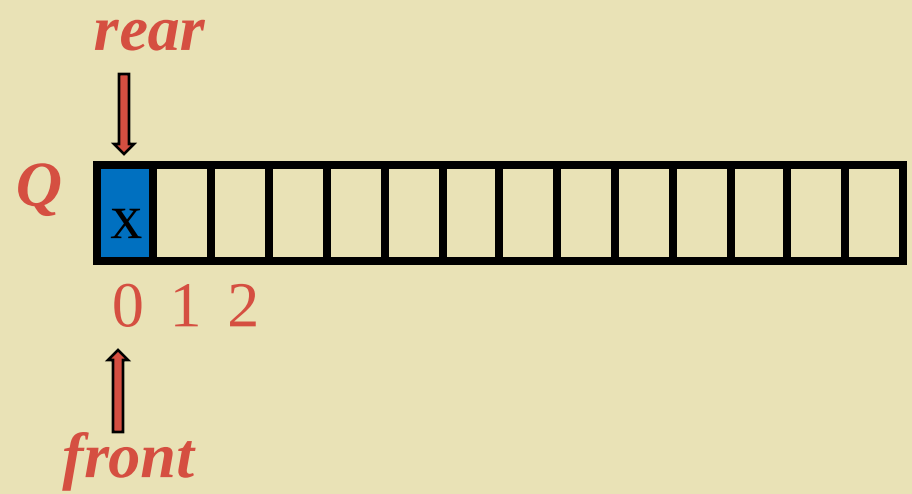
```
//Check if the Queue is full
IsFull () {
if rear == size(A) -1
        print "Queue is full"
        return true
else
        return false
}
```



Array-based Implementation

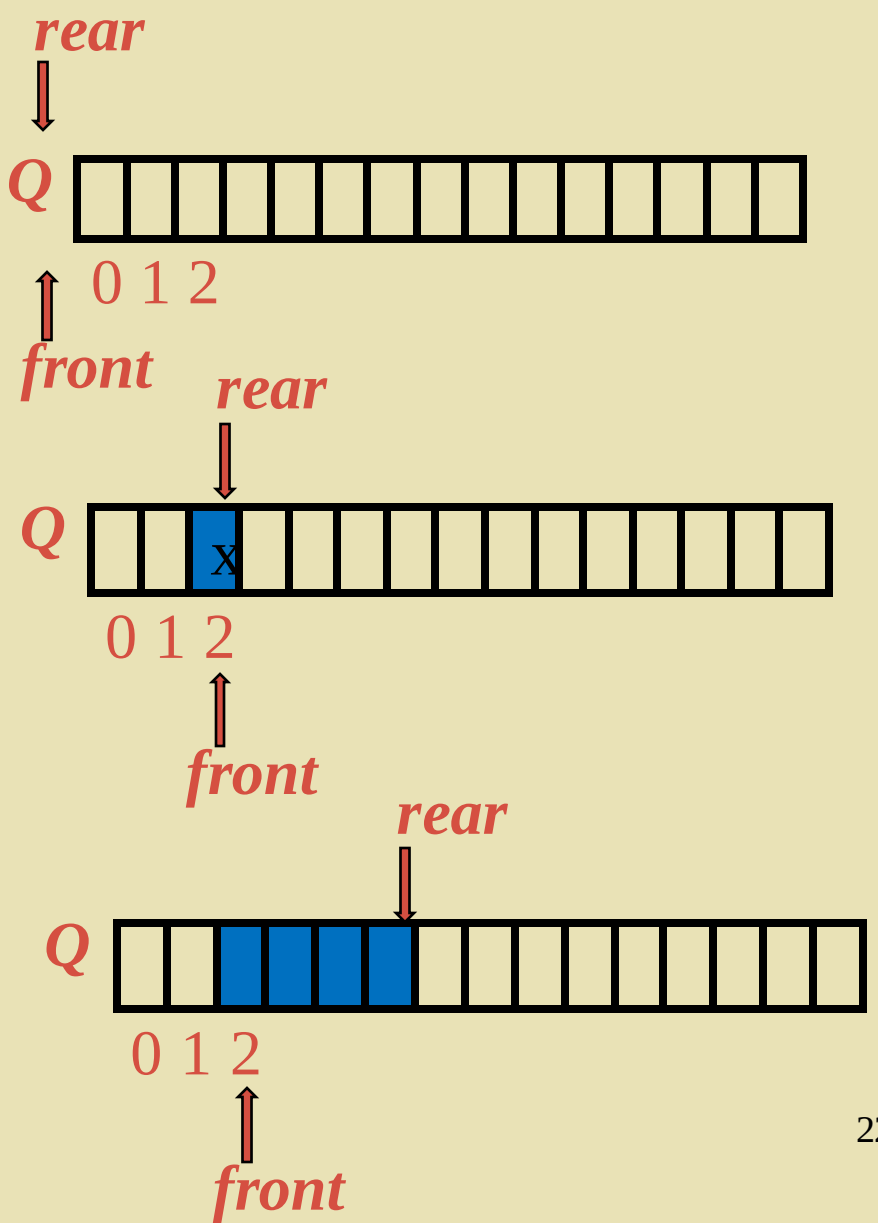
```
//Enqueue function
Enqueue (x) {
if IsFull()
        return
else if IsEmpty() {
        front = 0
        rear = 0
        }
    else {
        rear = rear + 1
        }
    A[rear] = x
}
```





Array-based Implementation

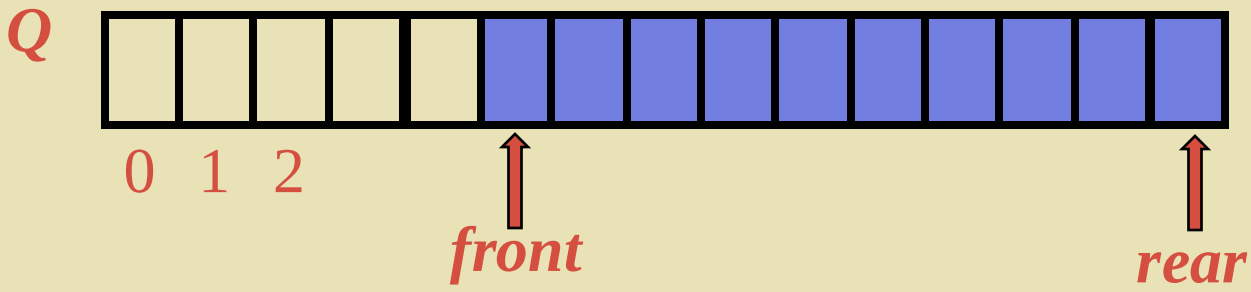
```
//Dequeue function
Dequeue () {
if IsEmpty()
        print "queue empty"
        return
else if front == rear {
// there is only one element
        Deq=A[front]
        front = -1
        rear = -1
        return Deq
        }
else front = front + 1
        return A[front-1]
}
```



Array-based Implementation

- Limitations
  - Queue size is bounded by the pre-defined array size
  - When *rear* reach the last index in the array, we cannot extend it further
  - There could be unused cells in the array

- Solutions?



Array-based Implementation

- Circular array
  - Current position is *i*
  - next position is *(i+1)%n*
  - previous position *(i+n-1)%n*

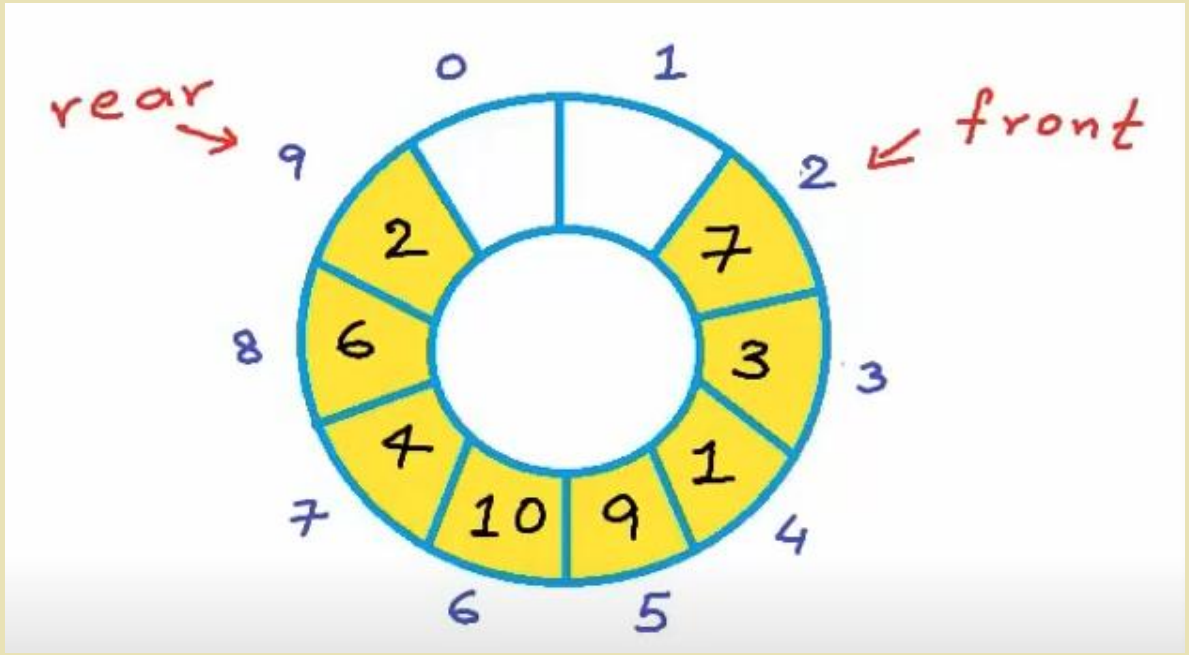


Figure from https://www.youtube.com/watch?v=okr-XE8yTO8

Array-based Implementation

```
//Enqueue function
Enqueue (x) {
if IsFull()
        return
else if IsEmpty() {
        front = 0
        rear = 0
        }
else {
        rear = rear + 1
        }
        A[rear] = x
}
```

```
//Enqueue circular array
Enqueue (x) {
if (rear+1)%n == front
        return
else if IsEmpty() {
        front = 0
        rear = 0
        }
else {
        rear = (rear + 1)%n
        }
        A[rear] = x
}
```

Note: Can change IsFull function

Array-based Implementation

```
//Dequeue function
Dequeue () {
if IsEmpty()
        print “queue empty”
        return
else if front == rear {
// there is only one element
        Deq=A[front]
        front = -1
        rear = -1
        return Deq
        }
else front = front + 1
        return A[front-1]
}
```

```
//Dequeue circular array
Dequeue () {
if IsEmpty()
        print “queue empty”
        return
else if front == rear {
// there is only one element
        Deq=A[front]
        front = -1
        rear = -1
        return Deq
        }
else front = (front + 1)%n
        return A[front-1]
}
```

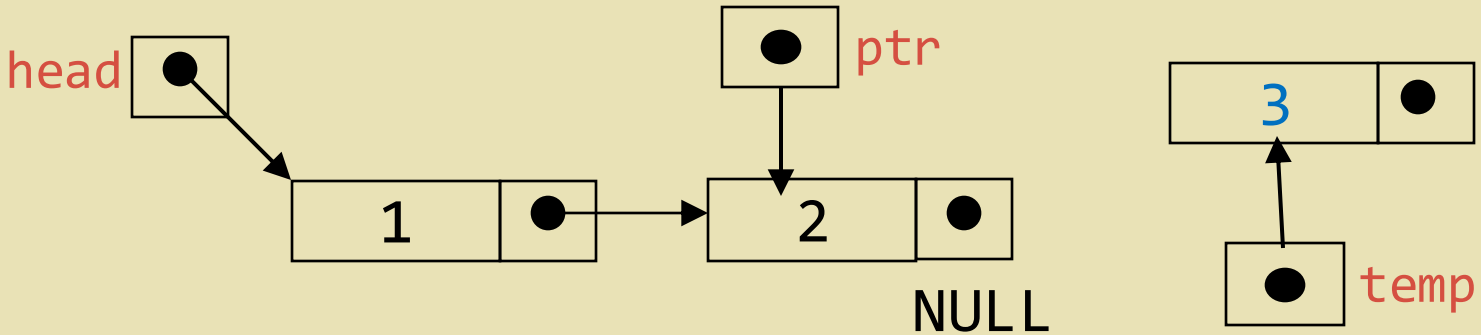
Linked-List Implementation

- LL's head node – queue's front
- LL's tail node – queue's rear
- Enqueue – append to the end of LL
- Dequeue – remove the head node

- Recall: how did we do that in the LL? Time complexity?

Inserting as a new last element

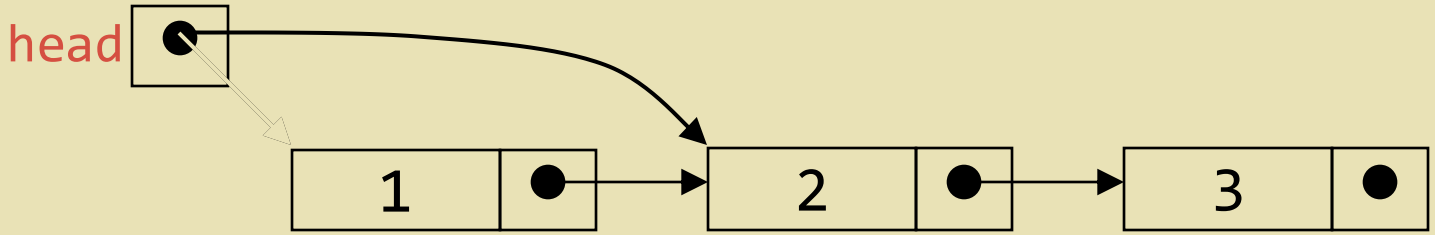
- How do we know where is the end? -- Traversal or Tail pointer




```
ptr -> next = temp;
```

Deleting an element from a SLL

- To delete the first element, change the head



```
// C program for linked list implementation of Queue
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>

// Node structure representing a single node in the linked list
typedef struct Node {
        int data;
        struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int new_data)
{
        Node* new_node = (Node*)malloc(sizeof(Node));
        new_node->data = new_data;
        new_node->next = NULL;
        return new_node;
}
```

Code from https://www.geeksforgeeks.org/queue-linked-list-implementation

```
// Structure to implement queue operations using a linked list
typedef struct Queue {
        Node *front, *rear;
} Queue;

// Function to create a queue
Queue* createQueue() {
        Queue* q = (Queue*)malloc(sizeof(Queue));
        q->front = NULL;
        q->rear = NULL;
        return q;
}

// Function to check if the queue is empty
int isEmpty(Queue* q)
{
// If the front and rear are null, then the queue is empty, otherwise it's not
        if (q->front == NULL && q->rear == NULL) {
                return 1;
        }
        return 0;
}
```

Code from https://www.geeksforgeeks.org/queue-linked-list-implementation

```
// Function to add an element to the queue
void enqueue(Queue* q, int new_data)
{
        // Create a new linked list node
        Node* new_node = createNode(new_data);
        // If queue is empty, the new node is both the front and rear
        if (q->rear == NULL)
        {
                q->front = new_node;
                 q->rear = new_node;
                return;
        }
        // Add the new node at the end of the queue and change rear
        q->rear->next = new_node;
        q->rear = new_node;
}
```

Code from https://www.geeksforgeeks.org/queue-linked-list-implementation

```
// Function to remove an element from the queue
void dequeue(Queue* q)
{
        // If queue is empty, return
        if (isEmpty(q))
        {
                printf("Queue Underflow\n");
                return;
        }
        // Store previous front and move front one node ahead
        Node* temp = q->front;
        q->front = q->front->next;

        // If front becomes null, then change rear also to null
        if (q->front == NULL)
                q->rear = NULL;
        // Deallocate memory of the old front node
        free(temp);
}
```

Code from https://www.geeksforgeeks.org/queue-linked-list-implementation

```
// Function to get the front element of the queue
int getFront(Queue* q)
{
        // Checking if the queue is empty
        if (isEmpty(q))
        {
                printf("Queue is empty\n");
                return INT_MIN;
        }
        return q->front->data;
}
```

```
// Function to get the rear element of the queue
int getRear(Queue* q)
{
        // Checking if the queue is empty
        if (isEmpty(q))
        {
                printf("Queue is empty\n");
                return INT_MIN;
        }
        return q->rear->data;
}
```

Code from https://www.geeksforgeeks.org/queue-linked-list-implementation

```
// Driver code
int main()
{
        Queue* q = createQueue();
        // Enqueue elements into the queue
        enqueue(q, 10);
        enqueue(q, 20);
        printf("Queue Front: %d\n", getFront(q));
        printf("Queue Rear: %d\n", getRear(q));
        // Dequeue elements from the queue
        dequeue(q);
        // Enqueue more elements into the queue
        enqueue(q, 30);
        enqueue(q, 40);
        // Dequeue an element from the queue
        dequeue(q);
        printf("Queue Front: %d\n", getFront(q));
        printf("Queue Rear: %d\n", getRear(q));
        return 0;
}
```

Code from https://www.geeksforgeeks.org/queue-linked-list-implementation

More Examples on Queue Implementations

- https://www.javatpoint.com/linked-list-implementation-of-queue
- https://www.scaler.com/topics/c/implementation-of-queue-using-linked-list/
- Data structures: Array implementation of Queue (https://www.youtube.com/watch?v=okr-XE8yTO8) and Linked List implementation of Queue (https://www.youtube.com/watch?v=A5_XdiK4J8A) by mycodeschool
- Linked list implementation of Queue in C
  - https://gist.github.com/mycodeschool/7510222
  - https://gist.github.com/kroggen/5fc7380d30615b2e70fcf9c7b69997b6 (no pointer tracking the *rear*)

Exercise

- If the head pointer is null, the queue _____.

A. is empty
B. is full
C. has at least one item

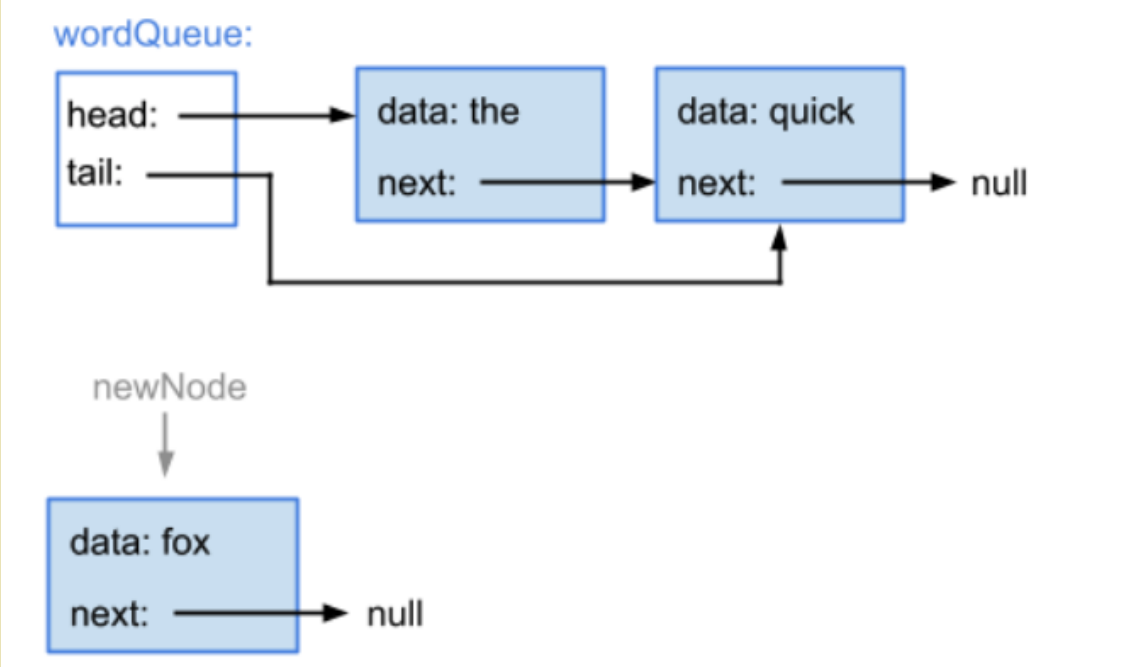
Exercise

- If the head pointer is null, the queue _____.

A. is empty
B. is full
C. has at least one item

Exercise

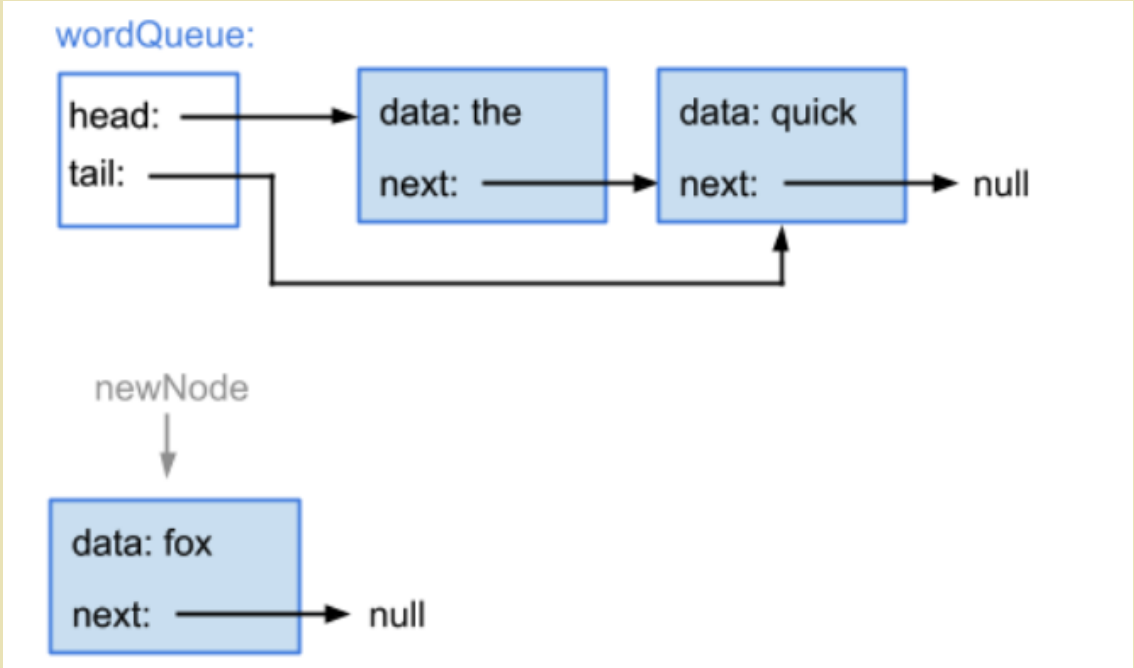
- For QueueEnqueue(wordQueue, "fox"), which pointer is updated to point to the node?



Exercise is from the textbook 5.5 Queues using linked lists

Exercise

- For QueueEnqueue(wordQueue, "fox"), which pointer is updated to point to the node?



The tail node's next pointer

Exercise is from the textbook 5.5 Queues using linked lists

References and Resources

- Course notes of ECE 250 Algorithms and Data Structures by Douglas W. Harder, University of Waterloo
- Data structures: Array implementation of Queue (https://www.youtube.com/watch?v=okr-XE8yTO8) and Linked List implementation of Queue (https://www.youtube.com/watch?v=A5_XdiK4J8A) by mycodeschool
- Linked list implementation of Queue in C
  - https://gist.github.com/mycodeschool/7510222
  - https://gist.github.com/kroggen/5fc7380d30615b2e70fcf9c7b69997b6 (no pointer tracking the *rear*)

That's about this lecture!