# Data Structures and Algorithms

**Prepared by: Mohamed Ayman**

Algorithm Engineer at Valeo

Deep Learning Researcher and Teaching Assistant

at The American University in Cairo (AUC)

**spring 2020**

sw.eng.MohamedAyman@gmail.com

linkedin.com/in/cs-MohamedAyman

github.com/cs-MohamedAyman

codeforces.com/profile/Mohamed_Ayman

# Lecture 5

# Queue

## Linked List Based

# Course Roadmap


## Part 1: Linear Data Structures

Lecture 1: Complexity Analysis and Recursion

Lecture 2: Arrays

Lecture 3: Linked List

Lecture 4: Stack

**Lecture 5: Queue**

Lecture 6: Deque

Lecture 7: STL in C++ (Linear Data Structures)

# Lecture Agenda

We will discuss in this lecture the following topics

1- Introduction to Queue

2- Insertion Operation

3- Deletion Operation

4- Front & Back Operations

5- Time Complexity & Space Complexity


Let's
START UP

# Lecture Agenda

Section 1: Introduction to Queue

Section 2: Insertion Operation

Section 3: Deletion Operation

Section 4: Front & Back Operations

Section 5: Time Complexity & Space Complexity

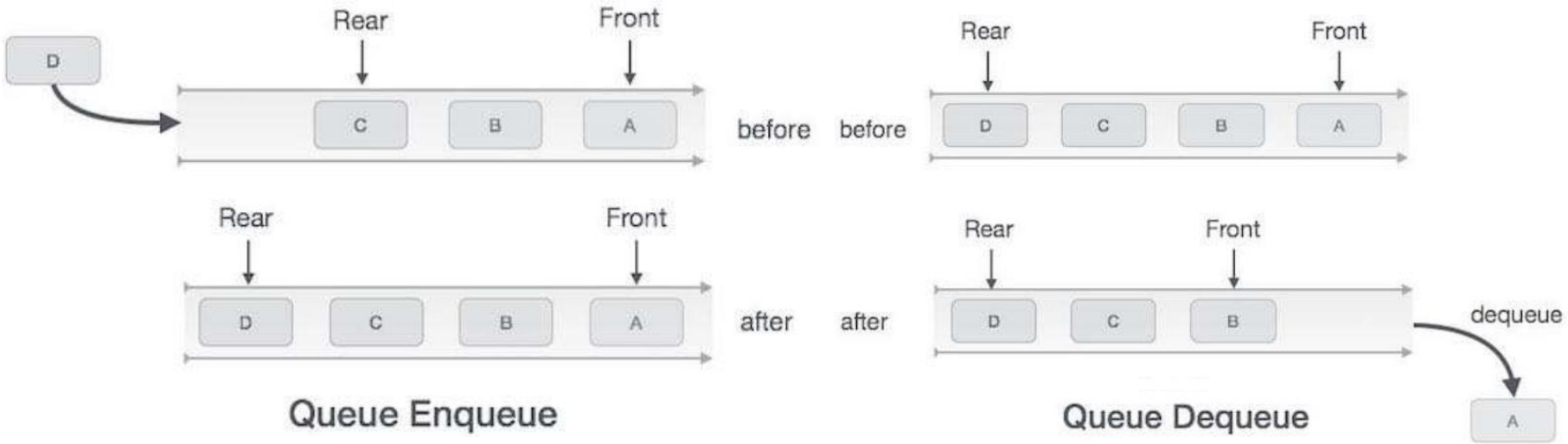
# Introduction to Queue

- A Queue is a linear structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.
- A queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence. By convention, the end of the sequence at which elements are added is called the back, tail, or rear of the queue, and the end at which elements are removed is called the head or front of the queue, analogously to the words used when people line up to wait for goods or services.
- The operations of adding an element to the rear of the queue is known as enqueue, and the operation of removing an element from the front is known as dequeue.

# Introduction to Queue

- Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues

- enqueue() - add (store) an item to the queue.
- dequeue() - remove (access) an item from the queue.



# Introduction to Queue

- Queue is also an abstract data type or a linear data structure, just like stack data structure, in which the first element is inserted from one end called the REAR(also called tail), and the removal of existing element takes place from the other end called as FRONT(also called head). This makes queue as FIFO (First in First Out) data structure, which means that element inserted first will be removed first.
- The operations of a queue make it a first-in-first-out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed. A queue is an example of a linear data structure, or more abstractly a sequential collection.

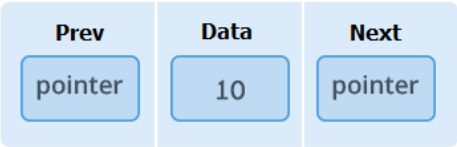
# Introduction to Queue

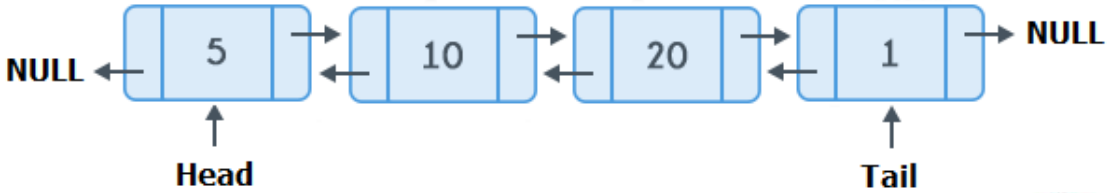
➢ Following are the basic operations supported by a queue.

- Push: which adds an element to the collection.
- Pop: which removes the most recently added element that was not yet removed.
- Front & Back: which gets the first element and the last element in the queue.

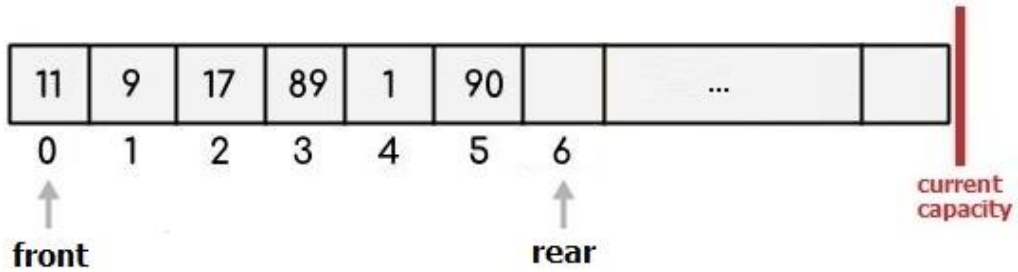
➢ Queue Types

1. Queue (Linked List Based)




2. Queue (Array Based)



# Queue (Linked List Based) Node

- Initialize a global struct

```
#include <bits/stdc++.h>
using namespace std;

// A queue node
struct node {
    int data;
    node* prev;
    node* next;
};

// Initialize a global pointers for head and tail
node* head;
node* tail;
```

Queue-Linked-List-Based.cpp

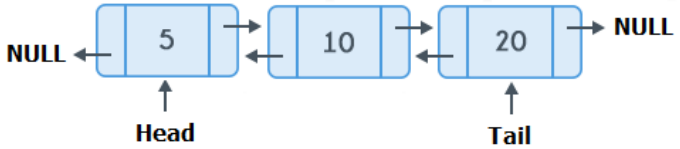
# Lecture Agenda

- ✓ Section 1: Introduction to Queue
- **Section 2: Insertion Operation**
- Section 3: Deletion Operation
- Section 4: Front & Back Operations
- Section 5: Time Complexity & Space Complexity

# Insertion Operation - Queue (Linked List Based)

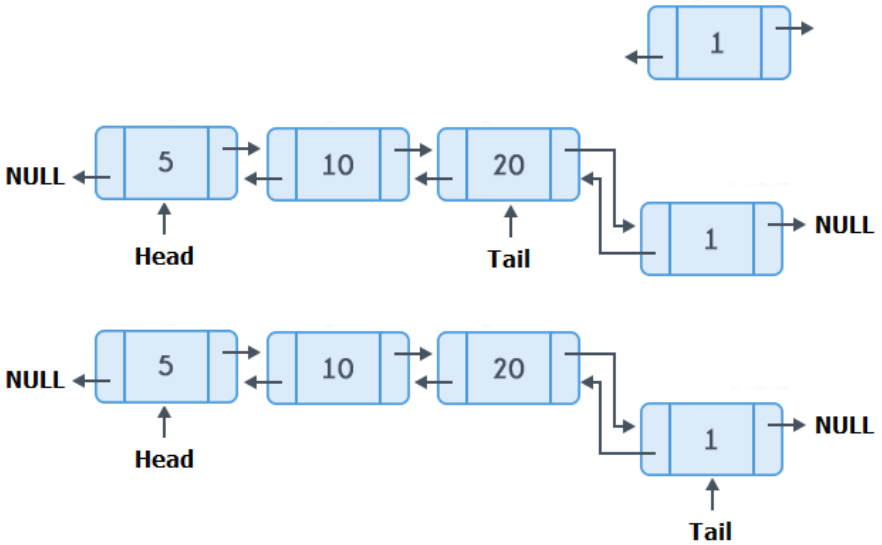
- Insert Operation is to add (store) an item to the queue.

- Insert 1




- ***Insertion Algorithm***:

1. ***create a new node***
2. ***new node data = data***
3. ***if head == NULL then head = new node***
4. ***tail = new node***
5. ***otherwise new node prev = tail***
6. ***tail next = new node***
7. ***tail = new node***



# Insertion Operation - Queue (Linked List Based)

```
// This function adds a node at the end of the queue
void push(int new_data) {
    // allocate new node and put it's data
    node* new_node = new node();
    new_node->data = new_data;
    // check if the queue is empty
    if (head == NULL) {
        head = new_node;
        tail = new_node;
    }
    // otherwise reach the end of the queue
    else {
        // set the next of the last node to be the new node and vice versa
        tail->next = new_node;
        new_node->prev = tail;
        // set the new node as a tail
        tail = new_node;
    }
}
```

Queue-Linked-List-Based.cpp

# Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- **Section 3: Deletion Operation**
- Section 4: Front & Back Operations
- Section 5: Time Complexity & Space Complexity

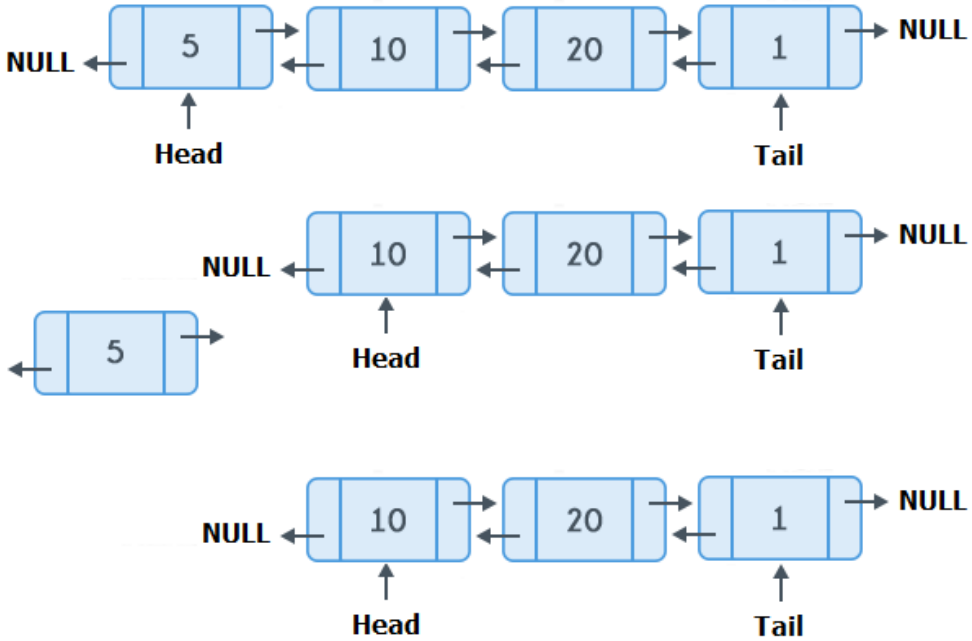
# Deletion Operation - Queue (Linked List Based)

- Delete Operation remove (access) an item from the queue.

- ***Deletion Algorithm:***

1. ***temp node = head***
2. ***if head equal tail then***
3. ***delete temp node***
4. ***head = tail = NULL***
5. ***otherwise head = head next***
6. ***head prev = NULL***
7. ***delete temp node***

- Delete 5



# Deletion Operation - Queue (Linked List Based)

```
// This function deletes the first node in the queue
void pop() {
    // check if the queue is empty
    if (head == NULL)
        return;
    // get the node which it will be deleted
    node* temp_node = head;
    // check if the queue has only one node
    if (head == tail) {
        delete(temp_node); // delete the temp node
        head = tail = NULL;
    }
    // otherwise the queue has nodes more than one
    else {
        // shift the head to be the next node
        head = head->next;
        head->prev = NULL;
        delete(temp_node); // delete the temp node
    }
}
```

Queue-Linked-List-Based.cpp

# Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- **Section 4: Front & Back Operations**
- Section 5: Time Complexity & Space Complexity

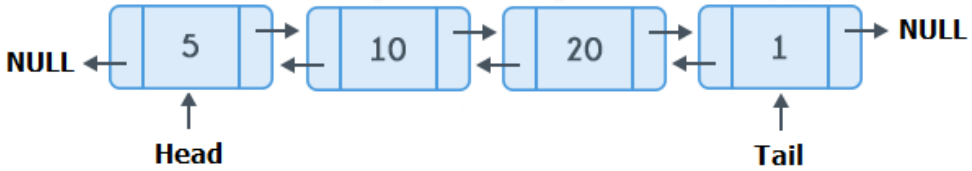
# Front & Back Operations - Queue (Linked List Based)

- Front Operation gets the first item in the queue.

- ***Front Algorithm***:

1. ***return head***

- Front 5

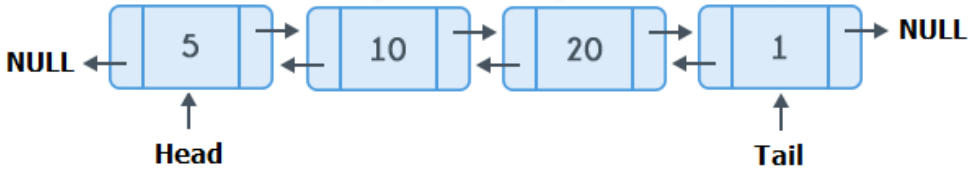



- Back Operation gets the last item in the queue.

- ***Back Algorithm***:

1. ***return tail***

- Back 1



# Front & Back Operations - Queue (Linked List Based)

```
// This function returns the value of the first node in the queue
int front() {
    // check if the queue is empty
    // to return the biggest integer value as an invalid value
    if (head == NULL)
        return INT_MAX;
    // otherwise return the real value
    else
        return head->data;
}
// This function returns the value of the last node in the queue
int back() {
    // check if the queue is empty
    // to return the biggest integer value as an invalid value
    if (tail == NULL)
        return INT_MAX;
    // otherwise return the real value
    else
        return tail->data;
}
```

Queue-Linked-List-Based.cpp

# Functionality Testing - Queue (Linked List Based)

- Initialize a global struct

```cpp
#include <bits/stdc++.h>
using namespace std;

// A queue node
struct node {
    int data;
    node* prev;
    node* next;
};

// Initialize a global pointers for head and tail
node* head;
node* tail;
```

- In the Main function:

```cpp
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
```

- Expected Output:

```
Queue front: 2147483647 and Queue back: 2147483647
```

Queue-Linked-List-Based.cpp

# Functionality Testing - Queue (Linked List Based)

- In the Main function:

```
push(10);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(20);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(30);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(40);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(50);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
```

- Expected Output:

```
Queue front: 10 and Queue back: 10
Queue front: 10 and Queue back: 20
Queue front: 10 and Queue back: 30
Queue front: 10 and Queue back: 40
Queue front: 10 and Queue back: 50
```

Queue-Linked-List-Based.cpp

# Functionality Testing - Queue (Linked List Based)

- In the Main function:

```
while (head != NULL) {
    cout << "Queue front: " << front()
         << " and Queue back: " << back() << '\n';
    pop();
    cout << "queue front has been deleted\n";
}
cout << "Queue is empty now\n";
```

- Expected Output:

```
Queue front: 10 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 20 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 30 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 40 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue front: 50 and Queue back: 50
queue front has been deleted
-----------------------------------
Queue is empty now
```

Queue-Linked-List-Based.cpp

# Functionality Testing - Queue (Linked List Based)

- In the Main function:

```
push(10);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(20);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
push(30);
cout << "Queue front: " << front() << " and Queue back: " << back() << '\n';
```

- Expected Output:

```
Queue front: 10 and Queue back: 10
Queue front: 10 and Queue back: 20
Queue front: 10 and Queue back: 30
```

Queue-Array-Based.cpp

# Functionality Testing - Queue (Linked List Based)

- In the Main function:

```
while (head != NULL) {
    cout << "Queue front: " << front()
         << " and Queue back: " << back() << '\n';
    pop();
    cout << "queue front has been deleted\n";
}
cout << "Queue is empty now\n";
```

- Expected Output:

```
Queue front: 10 and Queue back: 30
queue front has been deleted
-----------------------------------
Queue front: 20 and Queue back: 30
queue front has been deleted
-----------------------------------
Queue front: 30 and Queue back: 30
queue front has been deleted
-----------------------------------
Queue is empty now
```

Queue-Linked-List-Based.cpp

# Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations

**Section 5: Time Complexity & Space Complexity**

# Time Complexity & Space Complexity


➢ Time Analysis

| | Worst Case | Average Case |
|---|---|---|
| • Push | Θ(1) | θ(1) |
| • Pop | Θ(1) | θ(1) |
| • Front | Θ(1) | θ(1) |
| • Back | Θ(1) | θ(1) |

# Lecture Agenda

- ✓ Section 1: Introduction to Queue
- ✓ Section 2: Insertion Operation
- ✓ Section 3: Deletion Operation
- ✓ Section 4: Front & Back Operations
- ✓ Section 5: Time Complexity & Space Complexity

# Practice

# Practice


1- Sliding Window Maximum summation of all sub-arrays of size k
2- Check string is palindrome or not
3- Generate Binary Numbers from 1 to n
4- Reversing a queue using recursion
5- Reversing the first K elements of a Queue
6- Find the largest multiple of 3
7- Smallest multiple of a given number made of digits 0 and 9 only
8- Delete all elements in the queue
9- Sum of minimum and maximum elements of all subarrays of size k
10- First negative integer in every window of size k

# Assignment

# Implement STL Queue


- queues are a type of container adaptor, specifically designed to operate in a (first-in first-out) context, where elements are inserted into one end of the container and extracted from the other.
- queues are implemented as containers adaptors, which are classes that use an encapsulated object of a specific container class as its underlying container, providing a specific set of member functions to access its elements. Elements are pushed into the "back" of the specific container and popped from its "front".
- Queues are a type of container adaptors which operate in a first in first out type of arrangement. Elements are inserted at the back (end) and are deleted from the front.

More Info: cplusplus.com/reference/queue/queue/
More Info: en.cppreference.com/w/cpp/container/queue
More Info: geeksforgeeks.org/queue-cpp-stl/

# Implement STL Queue

- Member functions: (constructor) Construct queue (public member function)
  (empty) Test whether container is empty (public member function)
  (size) Return size (public member function)
  (front) Access next element (public member function)
  (back) Access last element (public member function)
  (push) Insert element (public member function)
  (pop) Remove top element (public member function)
  (swap) Swap contents (public member function)

More Info: cplusplus.com/reference/queue/queue/
More Info: en.cppreference.com/w/cpp/container/queue


DO
MORE.