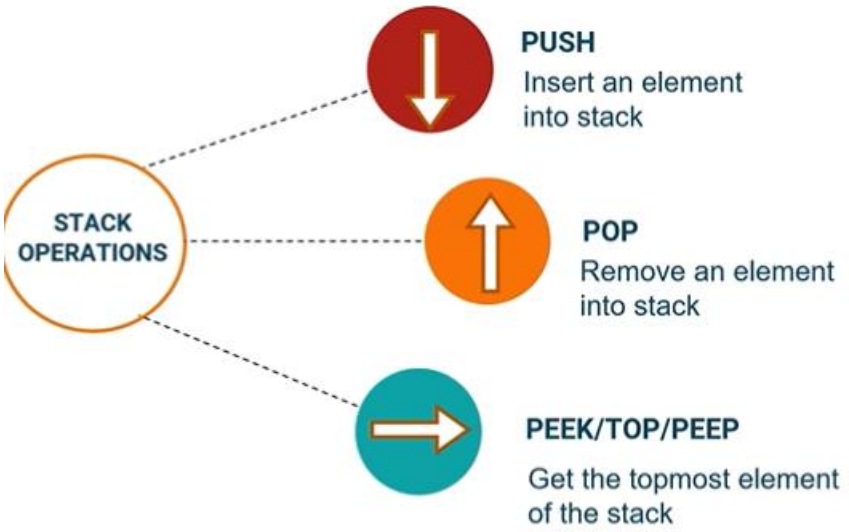
Linear Data Structure: Stack and Queue:

Stack is a pile of objects/items organized in a sequential manner. It is a linear data structure and it allows access only to the last element that is inserted and it works based on the principle of Last-In-First-Out (LIFO).

Stack operations:



Array Implementation of Stack

Insert an element into a stack: - push() operation

```
# define MAX 10
class Stack {
   int top;
   public:
     int arr[MAX];
     Stack() {
        top = -1;
     }
     void push(int item);
     int pop();
     int peek();
};
```

Defining 'Stack' class:

```
void Stack::push(int item) {
  if (top >= (MAX - 1)) {
     cout << "\nStack overflow";
  }
  else {
     arr[++top] = item;
     cout << "\n<< " Element added" "<< item;
   }
}
```

Remove an element from a stack :pop ()

```
int Stack::pop()
{
   if (top < 0) {
      cout << "\nStack underflow";
      return -1;
   }
   else {
      int item = arr[top--];
      return item;
   }
}
```

Retrieve the topmost element :peek()

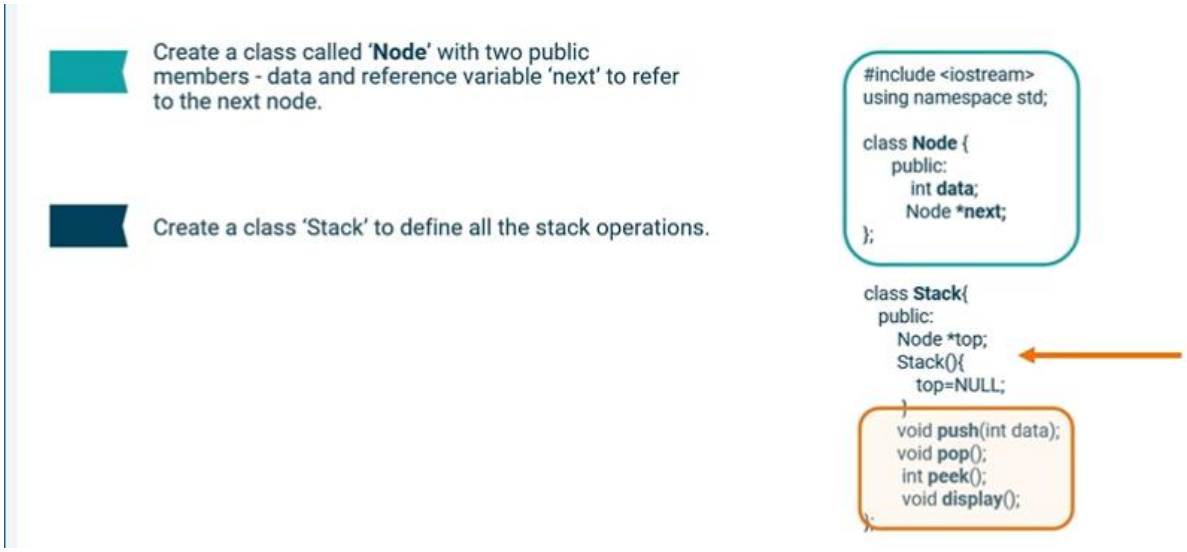
```
int Stack::peek()
{
   if (top < 0) {
      cout << "\nStack underflow";
      return -1;
   }
   else {
      int item = arr[top];
      return item;
   }
}
```

Driver Program

```
int main(){
    Stack st;
    st.push(8);
    st.push(4);
    st.push(5);
    st.push(9);
    int  value=st.pop();
    if (value!=-1)
      cout<<"\nDeleted element: "<<value;
    int top=st.peek();
    if(top!=-1)
       cout<<"\nTop element: " <<top;
    return 0;
}
```

Linked List of Implementation of Stack

1. Each node must contain at least 2 attributes: Data to be stored, and a reference to the next node.
2. Memory is allocated at the time of push operation and it will be released at the time of pop operation.
3. Stack elements can be inserted/deleted only from the top of the stack.



Insertion of an element / push operation:

Similar to inserting an element at the beginning of a linked list.

```
void Stack :: push(int n)
{
    Node *newNode=new Node();
    newNode->data=n;
    if(top == NULL) {
      newNode->next = NULL;
    }
    else {
      newNode->next = top;
    }
    top=newNode;
}
```

Deletion of an element / pop operation:

```
void Stack ::pop()
{
    Node* temp;
    if(top==NULL) {
      cout<<"Stack underflow"<<endl;
      return;
    }
   temp = top;
   top=top->next;
   delete temp;
}
```

Display the top of the stack (peek operation):

```
int Stack :: peek() {
      // Check for stack underflow
       if (top == NULL) {
           cout << "\nStack underflow";
           return -1; }
      return top->data;
}
```

Display the stack elements:

```
void Stack :: display() {
      Node *temp;
      // Check for stack underflow
      if (top == NULL) {
         cout << "\nStack underflow";
      }
      else {
         temp = top;
       while (temp != NULL) {
                cout << temp->data << " ";
                temp = temp->next;
        }
}}
```

Driver Program:

```
int main() {
  Stack stk;
  stk.push(30);
  stk.push(20);
  stk.push(50);
  stk.push(70);
   stk.display();
   stk.pop();
   int top_value=stk.peek();
   if (top_value!=-1){
       cout<<"\nTop value is :"<<
top_value<<"\n";
    }
    cout<<"Elements after the pop
operation:\n ";
    stk.display();
    return 0;
}
```

Application of Stack:

1. **Browsers**: Web browser use the stack to keep track of the web page-visited history.
2. **To Reverse a Word**: For reversing a string, it utilizes a stack data structure.
3. **Programming Language Processing**: Use stack to create local variables internally and to implement syntax checks for matching braces.
4. **Arithmetic Expression Evaluation**: Stack are used for evaluating the arithmetic expression used in a program.

Summary: Stack

- Stack is a Linear Data Structure that works based on the ***LIFO (Last In First Out)*** principle.
- In stack, all insertions and deletions are restricted to one end called **top** or **head**.
- Insertion of an item into the stack is known as a **push** operation.
- Deletion of an item from the stack is known as a **pop** operation.
- **Peek** refers to the retrieval of the topmost item in the stack.

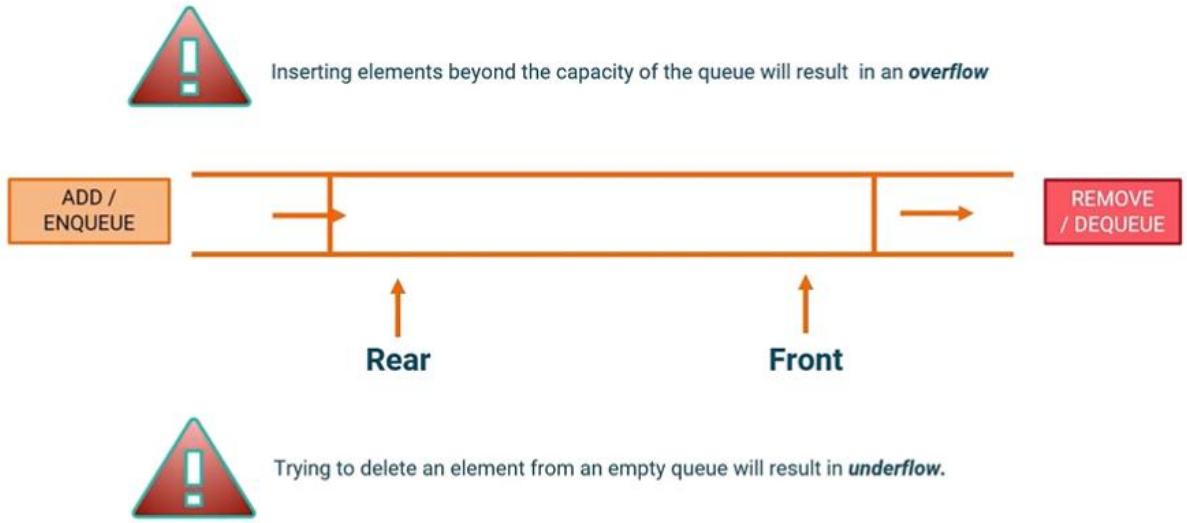
- Trying to insert elements beyond the capacity of a stack is known as **stack overflow.**
- Trying to delete an element from an empty stack is known as **stack underflow.**
- Two ways of implementing stack data structure: 1. Using **arrays.** 2. Using **Linked Lists.**

Queue is a linear data structure with restriction. Insertion happens at one end called **Rear** or **Tail** or **Back**. Deletion happens at one end called **Front** or **Head.** It works based on the **First In First Out (FIFO)** principle**.**

Applications of Queue:

1. CPU and disk scheduling
2. Handling of interruptions on real-time systems
3. Phone systems in call centers use queues to direct the calls in proper order
4. Shared resource usage (printer, memory access)

Queue Operations:



Queue Implementation using Array:

Creation of 'Queue' class:

```
#include <iostream>
using namespace std;
# define MAX 10

class Queue {
   int front, rear;
   int  arr[MAX];
   public:
   Queue() {
     front=rear=-1;
   }
      void enQueue(int item);
      void deQueue();
};
```

Initializing front and rear variables

Insert an element into a queue : enQueue()

```
void Queue::enQueue(int item)  {
    if(rear==MAX-1) {
          cout<<"\nQueue overflow";
    }
    else if(front==-1 && rear==-1) {
         front=rear=0;
         arr[rear]=item;
         cout<<"\nItem inserted "<<item;
     }
    else  {
         rear++;
        arr[rear]=item;
        cout<<"\nItem inserted "<<item;
  }
}
```

Delete an element from a queue : deQueue()

```
void Queue:: deQueue() {
   int item;
   if(rear==-1) {
       cout<<"\nQueue underflow";
   }
   else {
        item=arr[front];
        if(front==rear)
             front=rear=-1;

        else
              front++;
         cout<<"\nItem deleted  "<<item;
   }
}
```

Queue is empty

Driver Program

```
int main() {
   Queue q;
   q.enQueue(1);
   q.enQueue(2);
   q.enQueue(3);
   q.deQueue();
   q.deQueue();
   q.deQueue();
   q.deQueue();
   return 0;
}
```

Linked List Implementation of a Queue

Node creation:

```
#include <iostream>
using namespace std;

class Node {
   public:
     int data;
     Node *next;
};
```

Creation of 'Queue' class

```
class Queue{
  public:
     Node *front,*rear;
     Queue(){
        front=rear=NULL;
     }
     void enQueue(int n);
     void deQueue();
     void display();
};
```

ENQUEUE or INSERT Operation:

```
void Queue::enQueue(int n){
   Node *newNode=new Node();
   newNode->data=n;
   newNode->next=NULL;
   if(front==NULL){
     front=rear=newNode;
   }
   else{
     rear->next=newNode;
     rear=newNode;
   }
   cout<<n<<" Element inserted"<<endl;
}
```

DEQUEUE Operation:

```
void Queue :: deQueue()  {
   if (front==NULL){
      cout<<"Queue underflow"<<endl;
   }
  Node*tempPtr=front;
   if(front==rear)
     front=rear=NULL;
   else
      front=front->next;
    cout<<tempPtr->data<<" Element deleted
"<<endl;
   delete tempPtr;
}
```

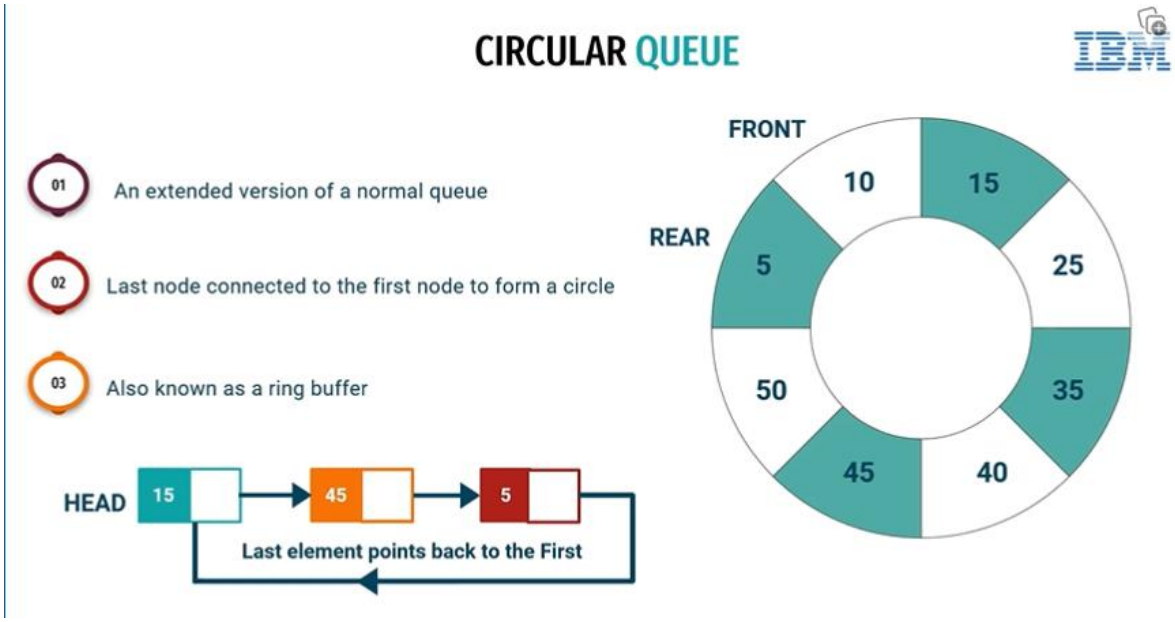
Display all the elements in the queue:

```
void Queue::display(){
      if (front==NULL){
        cout<<"Queue
 underflow"<<endl;
      }
       Node *temp=front;
      while(temp!=NULL) {
         cout<<temp->data<<" ";
        temp=temp->next;
      }
   }
```

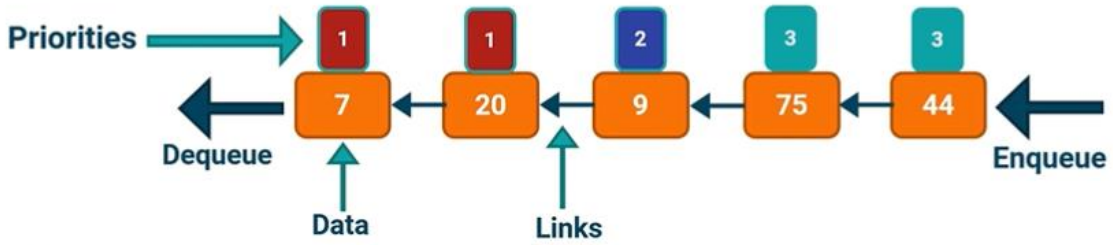
Driver program

```
int main() {
   Queue q;
   q.enQueue(10);
   q.enQueue(20);
   q.enQueue(30);
   q.enQueue(40);
   q.enQueue(50);
   q.deQueue();
   q.deQueue();
   q.display();
   return 0;
}
```

Types of Queue:



PRIORITY QUEUE



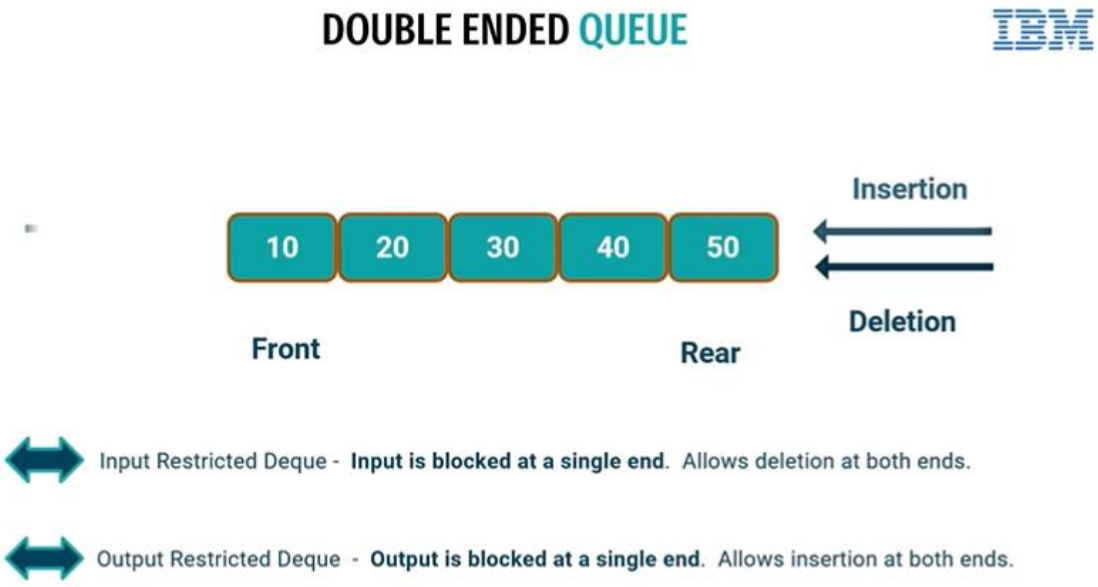
- A priority will be assigned to each element in the queue
- Elements are inserted at the rear of the queue, whereas elements are deleted based on the priority of the element.

PRIORITY QUEUE

- The element with the highest priority is removed first, and the element with the lowest priority is removed last.

DOUBLE ENDED QUEUE

- Insertion and deletion of an element can take place at **both ends**
- Also known as **Deque**



Summary: Queue

- A Queue is a linear data structure that works based on the **First In First Out (FIFO)** principle.
- An element can inserted into one end called the **REAR (tail/back),** and can be deleted from the other end called the **FRONT (head).**
- The process of adding an element into a queue at the rear end is called an **Enqueue** operation.
- The process of removing an element from the front of a queue is called as **Dequeue** operation.
- Trying to insert elements beyond the capacity of a queue results in a condition known as **overflow.**
- Trying to delete an element from an empty queue leads to a condition known as **underflow.**
- There are two ways to implement a queue data structure: Using an **array** and a **linked list.**
- In a circular queue, the **last node is connected back to the first node** to make a circle.
- In a priority queue, a priority is assigned to each element and these elements are **processed according to their priority.**
- Double ended queue **(deque)** allowed **insertion and deletion of an element at both ends.**