# INTRODUCTION

## What is a Stack ?

Data structure that holds a certain type of data which follows the LIFO mechanism.

LIFO : Last in First Out

Has 4 basic functions :
- Push : Add at end
- Pop : Remove from end
- Top : Return last element
- Size : Returns the size

Can be implemented using arrays or linked lists.

## What is a Queue ?

Data structure that holds a certain type of data which follows the FIFO mechanism.

FIFO : First in First Out

Has 4 basic functions :
- Push : Add at end
- Pop : Remove from start
- Top : Return last element
- Size : Returns the size

# Implementing a Stack using Arrays

We start with an array of maximum size and a variable top which starts at -1. As we push on pop elements, we increment or decrement it.

```
class Stack {
    int top;
    int stack[10];
    public:
    void push(int x) {
        stack[top + 1] = x;
        top += 1;
    }
    int pop() {
        if (top == -1) {
            printf("Stack is Empty");
            return -1;
        }
        top -= 1;
        return top;
    }
    int top() {
        if (top == -1) {
            printf("Stack is Empty");
            return -1;
        }
        return stack[top + 1];
    }
    int size() {
```

```
        return top+1;
    }
}
```

# Implementing a Queue using Arrays

```
class Queue {
    int start,end=-1,currentSize=0,size=10;
    int q[10];
    public:
    void push(int x) {
        if(currentSize < size) {
            if(start == -1) {
                q[start+1] = x;
                start += 1;
                end += 1;
            } else {
                q[end+1] = x;
                end = (end+1) % size;
            }
            currentSize += 1;
        } else {
            printf("Queue is full");
        }
    }
    int top() {
        if(start == -1) {
            printf("Queue is empty");
            return -1;
        }
        return q[start];
    }
    int pop() {
```

```
        if (start == -1) {
            printf("Stack is Empty");
            return -1;
        }
        if(currentSize == 1) {
            int x = q[end];
            start = -1;
            end = -1;
            return x;
        }
        int x = q[start];
        start = (start+1) % size;
        return x;
    }
}
```

Clearly, in both implementations, no search operation is going on, hence everything is O(1). But memory is static.

# Implementing Stacks using Linked Lists

At every push we track the end node and add one more.
Top operation is just returned with the last node.

```
class StackLL {
    Node * top;
    int size = 0;
    public:
```

```
    int push(int x) {
        Node * temp = new Node(x);
        temp -> next = top;
        top = temp;
        size += 1;
    }
    int top() {
        return top -> data;
    }
    int size() {
        return size;
    }
    int pop() {
        Node * temp = top;
        top = temp -> next;
        delete(temp);
        size -= 1;
    }
}
```

Everything is still O(1)

# Implementing a Queue using Linked List

```
class QueueLL {
    Node * start, end = NULL;
    int size = 0;
    public:
    int push(int x) {
        Node * temp = new Node(x);
        if (start == NULL) {
            start = temp;
```

```
            end = temp;
        } else {
            end -> next = temp;
            end = end -> next;
        }
        size += 1;
    }
    int pop() {
        if (start == NULL) {
            printf("Queue is empty");
            return -1;
        }
        Node * temp = start;
        start = start -> next;
        delete(temp);
        size -= 1;
    }
    int top() {
        if (start == NULL) {
            printf("Queue is Empty");
            return -1;
        }
        return end -> data;
    }
    int size() {
        return size;
    }
}
```

# Implement Stack using a Queue

For every push operation, we have to make sure the element is placed

at the start to maintain LIFO nature.

```
class StackFromQueue {
    queue<int> q;
    void push(x) {
        int s = q.size();
        q.push(x);
        for(int i=0; i<s; i++) {
            q.push(q.top());
            q.pop();
        }
    }
    int pop() {
        return q.pop();
    }
    int top() {
        return q.top();
    }
    int size() {
        return q.size();
    }
}
```

# Implementing a Queue using a Stack

Since Queue required tracking two values start and end but Stack only tracks one. So we use 2 stacks.

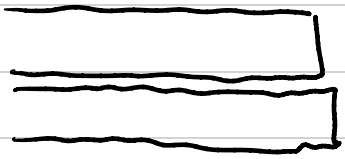
Procedure to Push :

① Everything from S1 to S2

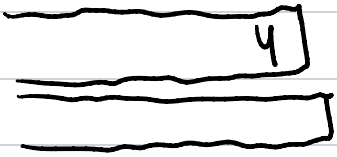
② Element to S1
③ Everything from S2 to S1

Eg : push(4)

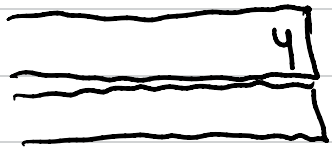
Step 1




Step 2

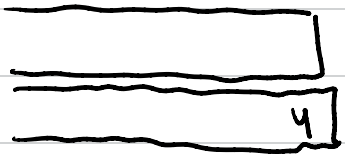



Step 3

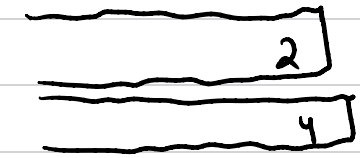



push(2)

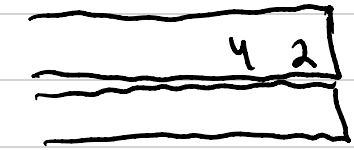
Step 1




Step 2

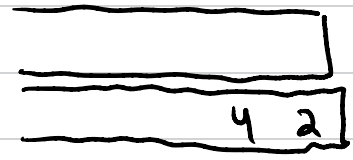



Step 3

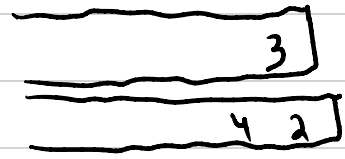



push(3)

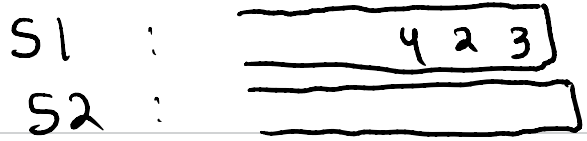
Step 1




Step 2




Step 3




and so on

```
class QueueUsingStack {
    stack <int> s1, s2;
    void push(int x) {
        while (s1.size() > 0) {
            s2.push(s1.top());
            s1.pop();
        }
        s1.push(x);
        while (s2.size() > 0) {
            s1.push(s2.top());
            s2.pop();
        }
    }
    int pop() {
        return s1.pop();
    }
    int top() {
        return s1.top();
    }
    int size() {
        return s1.size();
    }
}
```

Here push operation is $O(2N)$.
Previously it was $O(N)$.
Rest everything is $O(1)$.