Queue

```
#include <iostream>
using namespace std;
#define n 20
class queue
{
    int *arr;
    int front;
    int back;

public:
    queue()
    {
        arr = new int[n];
        front = -1; //
        back = -1; //
    }
    void push(int x)
    {
        if (back == n - 1)
        {
            cout << "Queue Overflow" << endl;
            return;
        }
        back++;
        arr[back] = x;
        if (front == -1)
        {
            front++;
        }
    }
    void pop()
    {
        if (front == -1 || front > back)
        {
            cout << "NOTHING TO POP" << endl;
            return;
        }
        front++;
    }
    int peek()
    {
        if (front == -1 || front > back)
        {
            cout << "NOTHING TO Peek" << endl;
            return -1;
        }
        return arr[front];
    }
    bool empty()
    {
        if (front == -1 || front > back)
        {
            cout << "Queue Empty" << endl;
            return true;
        }
        return false;
    }
       void display()
    {
        if (empty())
        {
            return;
        }
        for (int i = front; i <= back; i++)
        {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

int main()
{
    queue q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
// q.display();
   cout<<q.peek()<<endl;; /// 1
    q.pop();
    cout<<q.peek()<<endl;; /// 2
    q.pop();
    cout<<q.peek()<<endl; // 3
    q.pop();
   cout<<q.peek()<<endl; // 4
    return 0;
}
```

Queue using Linked List

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

class node{
    public:
    int data;
    node* next;
    node(int val){
        data=val;
        next=NULL;
    }
};
class queue{
    node* front;
    node* back;
    public:
    queue(){
        front=NULL;
        back=NULL;
    }
    void push(int x){
        node* n=new node(x);
        if(front==NULL){
            front=n;
            back=n;
        }
        back->next=n;
        back=n;
    }
    void pop(){
        if(front==NULL){
            cout<<"Queue Underflow"<<endl;
            return;
        }
        node* todelete=front;
        front=front->next;
        delete todelete;
    }

    int peek(){
        if(front==NULL){
            cout<<"Queue Underflow"<<endl;
            return -1;
        }
        return front->data;
    }

    bool isEmpty(){
        if(front==NULL){
            cout<<"Queue Empty";
            return true;
        }
        return false;
    }

};

int main(){
    queue q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(36);
    cout<<q.peek()<<endl;
    q.pop();
    cout<<q.peek()<<endl;
    q.pop();
    cout<<q.peek()<<endl;
    q.pop();
    cout<<q.isEmpty()<<endl;
cout<<q.peek()<<endl;
    q.pop();
    cout<<q.peek()<<endl;
    q.pop();
    q.isEmpty();

    return 0;
}
```

Queue using Stack

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

class queue{

    stack<int> s1;
    stack<int> s2;
    public:
    void push(int x){
        s1.push(x);
    }
    int pop(){
        if(s1.empty() && s2.empty()){
            cout<<"No Element to Pop"<<endl;
            return -1;
        }
        if(s2.empty()){
            while (!s1.empty())
            {
                s2.push(s1.top());
                s1.pop();
            }
        }
            int topval=s2.top();
            s2.pop();
            return topval;
    }
    bool empty(){
        if(s1.empty() && s2.empty()){
            return true;
        }
        return false;
    }

};

int main(){
    queue q;
    q.push(1);
    q.push(2);
    q.push(3);
    cout<<q.pop()<<endl;
    cout<<q.pop()<<endl;
    cout<<q.pop()<<endl;
}
```

Queue using stack with Recursion

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

class queue{
    stack<int> s1;
    public:
    void push(int x){
        s1.push(x);
    }
    int pop(){
        if(s1.empty()){
            cout<<"Queue Underflow"<<endl;
            return -1;
        }
       int x=s1.top();
       s1.pop();
       if(s1.empty()){
        // we are having only one element to pop
        return x;
       }
      //else recursively pop elements
      // pop will give us the answer we will again push rest of the elements and return the ans
      int item=pop();
      s1.push(x);
      return item;
    }
};

int main(){
    queue q;
    q.push(1);
    q.push(2);
    q.push(3);
    q.push(36);
    cout<<q.pop()<<endl;
    cout<<q.pop()<<endl;
    cout<<q.pop()<<endl;
    cout<<q.pop()<<endl;
    return 0;
}
```

Stack using Queue Method: Making push Costly

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

class Stack{
    queue<int> q1;
    queue<int> q2;
    int N;
    public:
    Stack(){
        N=0;
    }
    void Push(int x){
        /////// push it into q2
        q2.push(x);
        N++;
        ///// pop and push elements from q1 to q2
        while(!q1.empty()){
            q2.push(q1.front());
            q1.pop();

        }
        /// swap q1 and q2
        queue<int> temp=q1;
        q1=q2;
        q2=temp;
    }
    void pop(){
        q1.pop();// my last element is at the front of the q1 as by lifo of
stack we pop the last element
        N--;
    }
    int top(){
        return q1.front();
    }
    int size(){
        return N;
    }
};

int main(){
    Stack st;
    st.Push(1);
    st.Push(3);
    st.Push(36);
    cout<<st.top()<<endl;
    st.pop();
    cout<<st.top()<<endl;
    st.pop();
    cout<<st.size()<<endl;
    return 0;
}
```