Stack using Queue

A Stack is a **Last In First Out(LIFO)** structure, i.e, the element that is added last in the stack is taken out first. Our goal is to implement a Stack using Queue for which will be using two queues and design them in such a way that **pop** operation is same as dequeue but the **push** operation will be a little complex and more expensive too.

Implementation of Stack using Queue

Assuming we already have a class implemented for Queue, we first design the class for Stack. It will have the methods `push()` and `pop()` and two queues.

```
class Stack

{

    public:

    // two queue

    Queue Q1, Q2;

    // push method to add data element

    void push(int);

    // pop method to remove data element

    void pop();

};
```

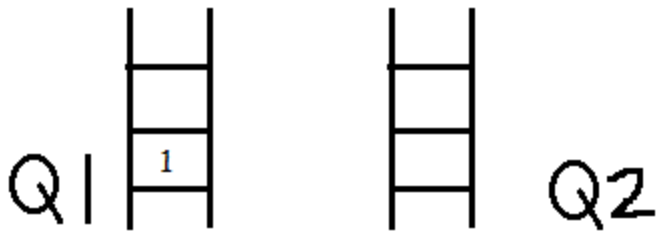
Copy

Inserting Data in Stack

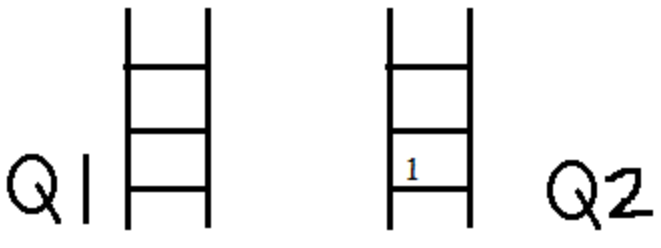
Since we are using Queue which is **First In First Out(FIFO)** structure , i.e, the element which is added first is taken out first, so we will implement the **push** operation in such a way that whenever there is a **pop** operation, the stack always pops out the last element added.

In order to do so, we will require two queues, `Q1` and `Q2`. Whenever the **push** operation is invoked, we will enqueue(move in this case) all the elements of `Q1` to `Q2` and then enqueue the new element to `Q1`. After this we will enqueue(move in this case) all the elements from `Q2` back to `Q1`.

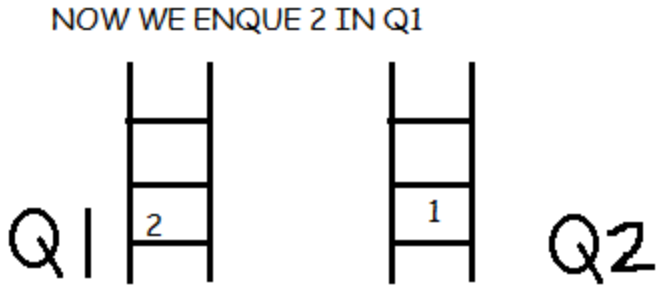
INITIALLY BOTH THE QUEUES ARE EMPTY.
WHEN WE GET A QUERY PUSH(1), SINCE Q1 IS EMPTY,
WE DIRECTLY INSERT 1 TO Q1.



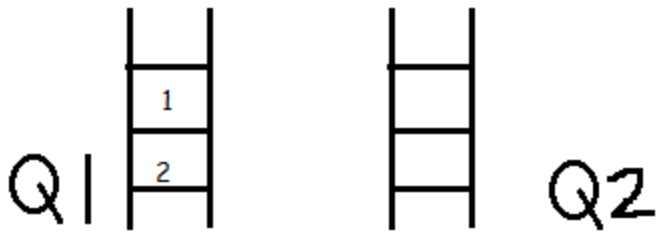
NOW IF WE GET A QUERY PUSH(2), WE WILL
FIRST ENQUE ALL ELEMENTS FROM Q1 TO Q2



NOW WE ENQUE 2 IN Q1



FINALLY WE DEQUE ALL THE ELEMENTS
FROM Q2 AND ENQUE THEM IN Q1



HENCE WE SEE THAT THE LAST ELEMENT ADDED , I.E., 2
IS IN THE FRONT OF Q1. SO IF WE GET A POP QUERY,
JUST USING THE DEQUE OPERATION IN Q1 WILL POP
OUT THE LAST ELEMENT ADDED IN THE STACK.

So let's implement this in our code,

```
void Stack :: push(int x)

{

     // move all elements in Q1 to Q2

     while(!Q1.isEmpty())

     {

          int temp = Q1.deque();

          Q2.enque(temp);

     }



     // add the element which is pushed into Stack

     Q1.enque(x);
```

```
        // move back all elements back to Q1 from Q2
        while(!Q2.isEmpty())
        {
              int temp = Q2.deque();
              Q1.enque(temp);
        }
}
```

Copy

It must be clear to you now, why we are using two queues. Actually the queue `Q2` is just for the purpose of keeping the data temporarily while operations are executed.

In this way we can ensure that whenever the **pop** operation is invoked, the stack always pops out the last element added in `Q1` queue.

Removing Data from Stack

Like we have discussed above, we just need to use the dequeue operation on our queue `Q1`. This will give us the last element added in Stack.

```
int Stack :: pop()
{
      return Q1.deque();
}
```

Copy

Time Complexity Analysis

When we implement Stack using a Queue the **push** operation becomes expensive.

- Push operation: O(n)
- Pop operation: O(1)

Conclusion

When we say "implementing Stack using Queue", we mean how we can make a Queue behave like a Stack, after all they are all logical entities. So for any data structure to act as a Stack, it should have `push()` method to add data on top and `pop()` method to remove data from top. Which

is exactly what we did and hence accomplished to make a Queue(in this case two Queues) behave as a Stack.
