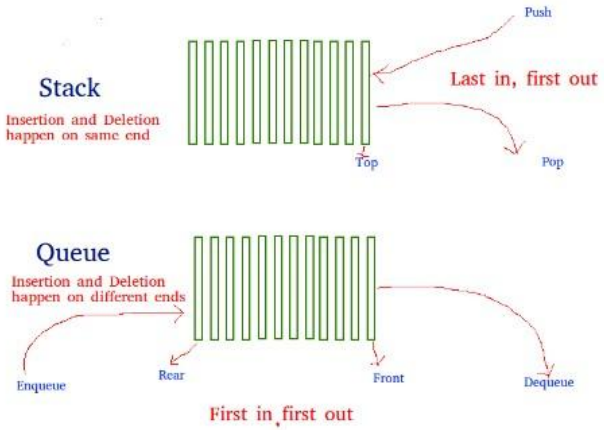
Implementing Queue using Stack

Problem: Given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on them.



Solution: *A queue can be implemented using two stacks.* Let the queue to be implemented be **q** and stacks used to implement **q** are **stack1** and **stack2** respectively.

The queue ***q*** can be implemented in two ways:

- **Method 1 (By making enQueue operation costly)**: This method makes sure that oldest entered element(element inserted first) is always at the top of stack1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used. The idea is to while pushing an element, first move all elements from stack1 to stack2, insert the new element to stack1 and then again move all elements from stack2 to stack1.

  Below is the implementation of both enQueue() and deQueue() operations:

```
enQueue(q, x)
  1) While stack1 is not empty, push everything from stack1 to stack2.
  2) Push x to stack1 (assuming size of stacks is unlimited).
  3) Push everything back to stack1.
Here the time complexity will be O(n)

deQueue(q)
  1) If stack1 is empty then print an error
  2) Pop an item from stack1 and return it
Here time complexity will be O(1)
```

Implementation:

C++ Java

```cpp
// CPP program to implement Queue using two stacks with costly enQueue()
#include <bits/stdc++.h>
using namespace std;
struct Queue {
    stack<int> s1, s2;
    void enQueue(int x)
    {
        // Move all elements from s1 to s2
        while (!s1.empty()) {
            s2.push(s1.top());
            s1.pop();
        }
        // Push item into s1
        s1.push(x);
        // Push everything back to s1
        while (!s2.empty()) {
            s1.push(s2.top());
            s2.pop();
        }
    }
    // Dequeue an item from the queue
    int deQueue()
    {
        // if first stack is empty
        if (s1.empty()) {
            cout << "Q is Empty";
            exit(0);
        }
        // Return top of s1
        int x = s1.top();
        s1.pop();
        return x;
    }
};

// Driver code
int main()
{
    Queue q;
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);

    cout << q.deQueue() << '\n';
    cout << q.deQueue() << '\n';
    cout << q.deQueue() << '\n';

    return 0;
}
```

Output:

```
1

2

3
```

Method 2 (By making deQueue operation costly): In this method, in en-queue operation, the new element is entered at the top of stack1. In de-queue operation, if stack2 is empty then all the elements are moved to stack2 and finally, the top of stack2 is returned.

Below is the implementation of both enQueue() and deQueue() operations:

```
enQueue(q,  x)
  1) Push x to stack1 (assuming size of stacks is unlimited).
Here time complexity will be O(1)

deQueue(q)
  1) If both stacks are empty then error.
  2) If stack2 is empty
       While stack1 is not empty, push everything from stack1 to stack2.
  3) Pop the element from stack2 and return it.
Here time complexity will be O(n)
```

Method 2 is better in performance than method 1. As Method 1 moves all the elements twice in enQueue operation, while method 2 (in deQueue operation) moves the elements once and moves elements only if stack2 is empty.

C++ Java

```cpp
// CPP program to implement Queue using two stacks with costly deQueue()
#include <bits/stdc++.h>
using namespace std;
struct Queue {
    stack<int> s1, s2;
    // Enqueue an item to the queue
    void enQueue(int x)
    {
        // Push item into the first stack
        s1.push(x);
    }
    // Dequeue an item from the queue
    int deQueue()
    {
        // if both stacks are empty
        if (s1.empty() && s2.empty()) {
            cout << "Q is empty";
            exit(0);
        }
        // if s2 is empty, move elements from s1
        if (s2.empty()) {
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        // return the top item from s2
        int x = s2.top();
        s2.pop();
        return x;
    }
};
// Driver code
int main()
{
    Queue q;
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);
    cout << q.deQueue() << '\n';
    cout << q.deQueue() << '\n';
    cout << q.deQueue() << '\n';
    return 0;
}
```

Output:

```
1 2 3
```