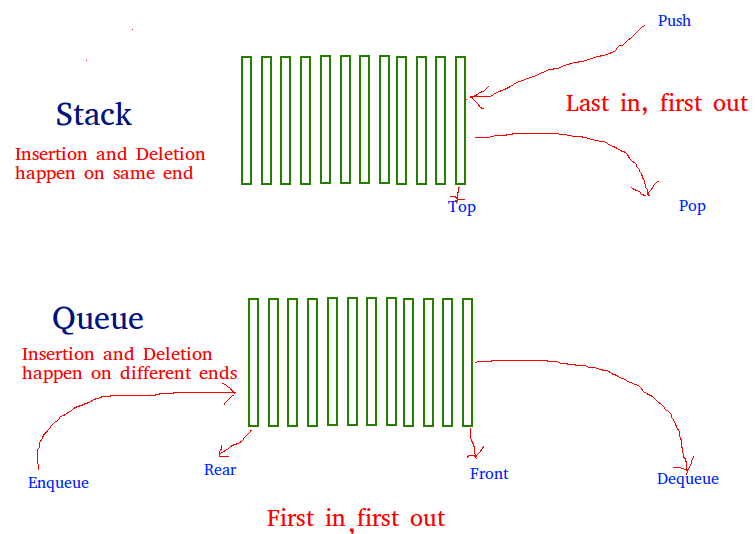
***Implementing Stack using Queue***

**Problem**: Given a Queue data structure that supports standard operations like enqueue() and dequeue(). We need to implement a Stack data structure using only instances of Queue and queue operations allowed on the instances.  
  
  
  
This problem is just the opposite of the problem described in the previous post of implementing a queue using stacks. Similar to the previous problem, a **stack**can also be implemented using two queues. Let stack to be implemented be **'s'** and queues used to implement be **'q1'** and **'q2'**.  
  
Stack 's' can be implemented in two ways:

* **Method 1 (By making push operation costly)**: This method makes sure that newly entered element is always at the front of 'q1', so that pop operation just dequeues from 'q1'. The queue, 'q2' is used to put every new element at front of 'q1'.
* **push(s, x)** // x is the element to be pushed and s is stack
* 1) Enqueue x to q2
* 2) One by one dequeue everything from q1 and enqueue to q2.
* 3) Swap the names of q1 and q2
* // Swapping of names is done to avoid one more
* // movement of all elements from q2 to q1.
* **pop(s)**
* 1) Dequeue an item from q1 and return it.

**Implementation**:C++Java

// Java Program to implement a stack using

// two queue

import java.util.\*;

class GfG

{

static class Stack

{

// Two inbuilt queues

static Queue<Integer> q1 = new LinkedList<Integer>();

static Queue<Integer> q2 = new LinkedList<Integer>();

// To maintain current number of

// elements

static int curr\_size;

Stack()

{

curr\_size = 0;

}

// Method to implement push() operation

static void push(int x)

{

curr\_size++;

// Push x first in empty q2

q2.add(x);

// Push all the remaining

// elements in q1 to q2.

while (!q1.isEmpty())

{

q2.add(q1.peek());

q1.remove();

}

// swap the names of two queues

Queue<Integer> q = q1;

q1 = q2;

q2 = q;

}

// Method to implement pop() operation

static void pop(){

// if no elements are there in q1

if (q1.isEmpty())

return ;

q1.remove();

curr\_size--;

}

// Method to get the top element of Stack

static int top()

{

if (q1.isEmpty())

return -1;

return q1.peek();

}

// Method to find the size of the Stack

static int size()

{

return curr\_size;

}

};

// Driver Code

public static void main(String[] args)

{

Stack s = new Stack();

s.push(1);

s.push(2);

s.push(3);

System.out.println("current size: " + s.size());

System.out.println(s.top());

s.pop();

System.out.println(s.top());

s.pop();

System.out.println(s.top());

System.out.println("current size: " + s.size());

}

}

**Output :**

current size: 3

3

2

1

current size: 1

* **Method 2 (By making pop operation costly)**: In push operation, the new element is always enqueued to q1. In pop() operation, if q2 is empty then all the elements except the last, are moved to q2. Finally the last element is dequeued from q1 and returned.
* **push(s, x)**
* 1) Enqueue x to q1 (assuming size of q1 is unlimited).
* **pop(s)**
* 1) One by one dequeue everything except the last element
* from q1 and enqueue to q2.
* 2) Dequeue the last item of q1, the dequeued item
* is the result, store it.
* 3) Swap the names of q1 and q2
* 4) Return the item stored in step 2.
* // Swapping of names is done to avoid one more
* // movement of all elements from q2 to q1.

**Implementation**:CPP

// Program to implement a stack using two queues

#include<bits/stdc++.h>

using namespace std;

// Stack class

class Stack

{

queue<int> q1, q2;

int curr\_size;

public:

Stack()

{

curr\_size = 0;

}

void pop()

{

if (q1.empty())

return;

// Leave one element in q1 and

// push others in q2.

while (q1.size() != 1)

{

q2.push(q1.front());

q1.pop();

}

// Pop the only left element

// from q1

q1.pop();

curr\_size--;

// swap the names of two queues

queue<int> q = q1;

q1 = q2;

q2 = q;

}

void push(int x)

{

q1.push(x);

curr\_size++;

}

int top()

{

if (q1.empty())

return -1;

while( q1.size() != 1 )

{

q2.push(q1.front());

q1.pop();

}

// last pushed element

int temp = q1.front();

// to empty the auxiliary queue after

// last operation

q1.pop();

// push last element to q2

q2.push(temp);

// swap the two queues names

queue<int> q = q1;

q1 = q2;

q2 = q;

return temp;

}

int size()

{

return curr\_size;

}

};

// Driver code

int main()

{

Stack s;

s.push(1);

s.push(2);

s.push(3);

s.push(4);

cout << "current size: " << s.size()

<< endl;

cout << s.top() << endl;

s.pop();

cout << s.top() << endl;

s.pop();

cout << s.top() << endl;

cout << "current size: " << s.size()

<< endl;

return 0;

}

**Output :**

current size: 4

4

3

2

current size: 2