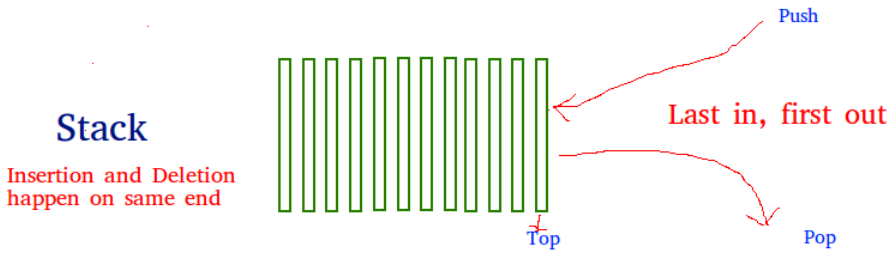
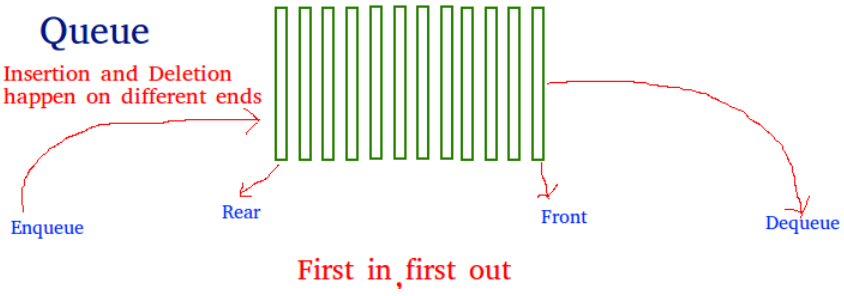
Implement Queue using Stacks

The problem is opposite of this post. We are 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.





A queue can be implemented using two stacks. Let queue to be implemented be q and stacks used to implement q be stack1 and stack2. q can be implemented in two ways:

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

Method 1 (By making enQueue operation costly) This method makes sure that oldest entered element is always at the top of stack 1, so that deQueue operation just pops from stack1. To put the element at top

of stack1, stack2 is used.

```
enQueue(q, x)
  1) While stack1 is not empty, push everything from satck1 to stack2.
  2) Push x to stack1 (assuming size of stacks is unlimited).
  3) Push everything back to stack1.

dnQueue(q)
  1) If stack1 is empty then error
  2) Pop an item from stack1 and return it
```

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 top of stack2 is returned.

```
enQueue(q,  x)
  1) Push x to stack1 (assuming size of stacks is unlimited).

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.
```

Method 2 is definitely better than method 1.
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 empty.
Implementation of method 2:

C

```c
/* Program to implement a queue using two stacks */
#include<stdio.h>
#include<stdlib.h>

/* structure of a stack node */
struct sNode
{
    int data;
    struct sNode *next;
};

/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data);

/* Function to pop an item from stack*/
int pop(struct sNode** top_ref);

/* structure of queue having two stacks */
struct queue
{
    struct sNode *stack1;
    struct sNode *stack2;
};
```

```
/* Function to enqueue an item to queue */
void enQueue(struct queue *q, int x)
{
    push(&q->stack1, x);
}

/* Function to dequeue an item from queue */
int deQueue(struct queue *q)
{
    int x;
    /* If both stacks are empty then error */
    if(q->stack1 == NULL && q->stack2 == NULL)
    {
        printf("Q is empty");
        getchar();
        exit(0);
    }

/* Move elements from satck1 to stack 2 only if
stack2 is empty */
if(q->stack2 == NULL)
{
    while(q->stack1 != NULL)
    {
        x = pop(&q->stack1);
        push(&q->stack2, x);

    }
}

x = pop(&q->stack2);
return x;
}

/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data)
{
    /* allocate node */
    struct sNode* new_node =
        (struct sNode*) malloc(sizeof(struct sNode));
        if(new_node == NULL)
        {
            printf("Stack overflow \n");
            getchar();
            exit(0);

        }

/* put in the data */
new_node->data = new_data;

/* link the old list off the new node */
new_node->next = (*top_ref);

/* move the head to point to the new node */
(*top_ref) = new_node;
}

/* Function to pop an item from stack*/
int pop(struct sNode** top_ref)
{
    int res;
    struct sNode *top;

    /*If stack is empty then error */
    if(*top_ref == NULL)
    {
        printf("Stack overflow \n");
        getchar();
        exit(0);
```

```
    }
    else
    {
        top = *top_ref;
        res = top->data;
        *top_ref = top->next;
        free(top);
        return res;

    }
}

/* Driver function to test anove functions */
int main()
{
    /* Create a queue with items 1 2 3*/
    struct queue *q = (struct queue*)malloc(sizeof(struct queue));
    q->stack1 = NULL;
    q->stack2 = NULL;
    enQueue(q, 1);
    enQueue(q, 2);
    enQueue(q, 3);

    /* Dequeue items */
    printf("%d ", deQueue(q));
    printf("%d ", deQueue(q));
    printf("%d ", deQueue(q));

getchar();
}
```

Run on IDE

Java

```
/* Java Program to implement a queue using two stacks */
// Note that Stack class is used for Stack implementation

import java.util.Stack;

public class GFG
{
    /* class of queue having two stacks */
    static class Queue
    {
        Stack<Integer> stack1 ;
        Stack<Integer> stack2 ;
    }

    /* Function to push an item to stack*/
    static void push(Stack<Integer> top_ref, int new_data)
    {
        //Push the data onto the stack
        top_ref.push(new_data);
    }

    /* Function to pop an item from stack*/
    static int pop(Stack<Integer> top_ref)
    {
        /*If stack is empty then error */
        if(top_ref.isEmpty())
        {
            System.out.println("Stack Overflow");
            System.exit(0);
        }
        //pop the data from the stack
```

```
        return top_ref.pop();
    }
    //Function to enqueue an item to the queue
    static void enQueue(Queue q, int x)
    {
        push(q.stack1, x);
    }
    /* Function to dequeue an item from queue */
    static int deQueue(Queue q)
    {
        int x;
        /* If both stacks are empty then error */
        if(q.stack1.isEmpty() && q.stack2.isEmpty() )
        {
            System.out.println("Q is empty");
            System.exit(0);
        }

        /* Move elements from stack1 to stack 2 only if
        stack2 is empty */
        if(q.stack2.isEmpty())
        {
            while(!q.stack1.isEmpty())
            {
            x = pop(q.stack1);
            push(q.stack2, x);

            }
        }
        x = pop(q.stack2);
        return x;
    }

    /* Driver function to test anove functions */
    public static void main(String args[])
    {
        /* Create a queue with items 1 2 3*/
        Queue q= new Queue();
        q.stack1 = new Stack<>();
        q.stack2 = new Stack<>();
        enQueue(q, 1);
        enQueue(q, 2);
        enQueue(q, 3);

        /* Dequeue items */
        System.out.print(deQueue(q)+" ");
        System.out.print(deQueue(q)+" ");
        System.out.println(deQueue(q)+" ");
    }
}
//This code is contributed by Sumit Ghosh
```

Run on IDE

Output:

```
1 2 3
```

Queue can also be implemented using one user stack and one Function Call Stack. Below is modified Method 2 where recursion (or Function Call Stack) is used to implement queue using only one user defined stack.

```
enQueue(x)
  1) Push x to stack1.

deQueue:
  1) If stack1 is empty then error.
  2) If stack1 has only one element then return it.
  3) Recursively pop everything from the stack1, store the popped item
    in a variable res,  push the res back to stack1 and return res
```

The step 3 makes sure that the last popped item is always returned and since the recursion stops when there is only one item in *stack1* (step 2), we get the last element of *stack1* in dequeue() and all other items are pushed back in step

3. Implementation of method 2 using Function Call Stack:

C

```
/* Program to implement a queue using one user defined stack
and one Function Call Stack */
#include<stdio.h>
#include<stdlib.h>

/* structure of a stack node */
struct sNode
{
    int data;
    struct sNode *next;
};

/* structure of queue having two stacks */
struct queue
{
    struct sNode *stack1;
};

/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data);

/* Function to pop an item from stack*/
int pop(struct sNode** top_ref);

/* Function to enqueue an item to queue */
void enQueue(struct queue *q, int x)
{
    push(&q->stack1, x);
}

/* Function to dequeue an item from queue */
int deQueue(struct queue *q)
{
    int x, res;

    /* If both stacks are empty then error */
    if(q->stack1 == NULL)
    {
        printf("Q is empty");
        getchar();
        exit(0);

    }
    else if(q->stack1->next == NULL)
    {
```

```
        return pop(&q->stack1);

    }
    else
    {
        /* pop an item from the stack1 */
        x = pop(&q->stack1);

        /* store the last dequeued item */
        res = deQueue(q);

        /* push everything back to stack1 */
        push(&q->stack1, x);
        return res;

    }
}

/* Function to push an item to stack*/
void push(struct sNode** top_ref, int new_data)
{
    /* allocate node */
    struct sNode* new_node =
            (struct sNode*) malloc(sizeof(struct sNode));

    if(new_node == NULL)
    {
        printf("Stack overflow \n");
        getchar();
        exit(0);

    }

/* put in the data */
new_node->data = new_data;

/* link the old list off the new node */
new_node->next = (*top_ref);

/* move the head to point to the new node */
(*top_ref) = new_node;
}

/* Function to pop an item from stack*/
int pop(struct sNode** top_ref)
{
    int res;
    struct sNode *top;

    /*If stack is empty then error */
    if(*top_ref == NULL)
    {
        printf("Stack overflow \n");
        getchar();
        exit(0);

    }
    else
    {
    top = *top_ref;
    res = top->data;
    *top_ref = top->next;
    free(top);
    return res;
    }
}

/* Driver function to test above functions */
int main()
{
```

```
    /* Create a queue with items 1 2 3*/
    struct queue *q = (struct queue*)malloc(sizeof(struct queue));
    q->stack1 = NULL;

    enQueue(q, 1);
    enQueue(q, 2);
    enQueue(q, 3);

    /* Dequeue items */
    printf("%d ", deQueue(q));
    printf("%d ", deQueue(q));
    printf("%d ", deQueue(q));

getchar();
}
```

Run on IDE

Java

```
// Java Program to implement a queue using one stack

import java.util.Stack;

public class QOneStack
{
    //class of queue having two stacks
    static class Queue
    {
        Stack<Integer> stack1;
    }

    /* Function to push an item to stack*/
    static void push(Stack<Integer> top_ref,int new_data)
    {
        /* put in the data */
        top_ref.push(new_data);
    }

    /* Function to pop an item from stack*/
    static int pop(Stack<Integer> top_ref)
    {
        /*If stack is empty then error */
        if(top_ref == null)
        {
            System.out.println("Stack Overflow");
            System.exit(0);
        }
        //return element from stack
        return top_ref.pop();
    }

    /* Function to enqueue an item to queue */
    static void enQueue(Queue q,int x)
    {
        push(q.stack1,x);
    }

    /* Function to dequeue an item from queue */
    static int deQueue(Queue q)
    {
        int x,res=0;
        /* If the stacks is empty then error */
        if(q.stack1.isEmpty())
        {
            System.out.println("Q is Empty");
            System.exit(0);
```

```
        }
        //Check if it is a last element of stack
        else if(q.stack1.size() == 1)
        {
            return pop(q.stack1);
        }
        else
        {
            /* pop an item from the stack1 */
            x=pop(q.stack1);

            /* store the last dequeued item */
            res = deQueue(q);

            /* push everything back to stack1 */
            push(q.stack1,x);
            return res;
        }
        return 0;
    }

    /* Driver function to test above functions */
    public static void main(String[] args)
    {
        /* Create a queue with items 1 2 3*/
        Queue q = new Queue();
        q.stack1 = new Stack<>();

        enQueue(q, 1);
        enQueue(q, 2);
        enQueue(q, 3);

        /* Dequeue items */
        System.out.print(deQueue(q) + " ");
        System.out.print(deQueue(q) + " ");
        System.out.print(deQueue(q) + " ");

    }

}
//This code is contributed by Sumit Ghosh
```

Run on IDE

Output:

```
1 2 3
```

Asked in: Inmobi, Accolite, Adobe, Amazon, DE Shaw, Flipkart, Goldman Sachs, InfoEdge, MakeMyTrip, Microsoft, Oracle

Please write comments if you find any of the above codes/algorithms incorrect, or find better ways to solve the same problem.

Queue Stack

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Implement Stack using Queues
Check whether a given Binary Tree is Complete or not | Set 1 (Iterative Solution)
Design and Implement Special Stack Data Structure | Added Space Optimized Version
Find the first circular tour that visits all petrol pumps
LRU Cache Implementation
Reversing a queue using recursion
Smallest multiple of a given number made of digits 0 and 9 only
Sorting a Queue without extra space
Implement PriorityQueue through Comparator in Java
Sharing a queue among three threads

(Login to Rate)

2.5 Average Difficulty : **2.5/5.0**
Based on **203** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE

Write an Article
GBlog
Videos