**Godavari College Of Engineering, Jalgaon.**

**Subject Name:** Data Structure. **Teacher Name:** Prof. S.S.Shete

**Practical No. :**  8 **Date:**

**Class:** S.E **Roll No:**

**Title:** Write a program to implement a queue using two stacks such that the enqueue operation runs in constant time and dequeue operation runs in linear time.

**Aim:** To implements the queue using two stacks such that the enqueue opration runs in constant time and dequeue opration runs in linear time.

**Theory:**

**Queue:**

A queue is a data structure for storing data where the order in which data arrives is important. The queue is implemented as FIFO (First in First Out).

The queue is an ordered list in which insertions are done at one end (rear) and deletions are done from another end (front). The first element that got inserted is the first one to be deleted (basic principle of FIFO).

The main Queue operations are:

1. **EnQueue (int data):** Insertion at rear end
2. **int DeQueue():** Deletion from front end

Stack:

A stack is also another data structure which is implemented as LIFO.

The stack is an ordered list where insertion and deletion are done from the same end, top. The last element that entered first is the first one to be deleted (the basic principle behind the LIFO).

The main stack operations are:

1. **push (int data):** Insertion at top
2. **int pop():** Deletion from top

**Implementing Queue using two stack :-**

**Algorithem :-**

A) Enqueue operation:1. Simply push the elements into the first stack.

B)Dequeue operation:1. Pop from the second stack if the second stack is not empty.2. If second stack is empty, pop from the first stack and push all the elements into second until  , the first stack becomes empty.3. Now pop an element from the second stack.

Program:

#include <stdio.h>

#include <stdlib.h>

int max=5;

struct node

{

int data;

struct node \*next;

};

void push(struct node\*\* top, int data);

int pop(struct node\*\* top);

struct queue

{

struct node \*stack1;

struct node \*stack2;

};

void enqueue(struct queue \*q, int x)

{

push(&q->stack1, x);

}

void dequeue(struct queue \*q)

{

int x;

if (q->stack1 == NULL && q->stack2 == NULL) {

printf("queue is empty");

return;

}

if (q->stack2 == NULL) {

while (q->stack1 != NULL) {

x = pop(&q->stack1);

push(&q->stack2, x);

}

}

x = pop(&q->stack2);

printf("%d\n", x);

}

void push(struct node\*\* top, int data)

{

struct node\* newnode = (struct node\*) malloc(sizeof(struct node));

if (newnode == NULL)

{

printf("Stack overflow \n");

return;

}

newnode->data = data;

newnode->next = (\*top);

(\*top) = newnode;

}

int pop(struct node\*\* top)

{

int buff;

struct node \*t;

if (\*top == NULL) {

printf("Stack underflow \n");

return 0;

}

else {

t = \*top;

buff = t->data;

\*top = t->next;

free(t);

return buff;

}

}

void display(struct node \*top1,struct node \*top2)

{

while (top1 != NULL) {

printf("%d\n", top1->data);

top1 = top1->next;

}

while (top2 != NULL) {

printf("%d\n", top2->data);

top2 = top2->next;

}

}

int main()

{

struct queue \*q = (struct queue\*)malloc(sizeof(struct queue));

int f = 0, a;

//char ch = 'y';

q->stack1 = NULL;

q->stack2 = NULL;

printf("\n\n\t 1.Insert Element in queue. \n\t 2.Remove Element from queue \n\t 3.Display \n\t 4.Exit\n");

while (f!=4)

{

printf("\n\n\t Enter Your Choice :- ");

scanf("%d", &f);

switch(f)

{

case 1 : printf("Enter the element to be added to queue :- ");

scanf("%d", &a);

enqueue(q, a);

printf("\n========================================================");

break;

case 2 : dequeue(q);

printf("\n========================================================");

break;

case 3 : display(q->stack1, q->stack2);

printf("\n========================================================");

break;

case 4 : printf("\n\n\t Thank you for visiting Program.");

printf("\n\t You Are Exited From Program.");

exit(1);

printf("\n========================================================");

break;

default : printf("invalid\n");

break;

}

}

return 0;

}

**Output**:



**Conclusion:**