## QUESTION 1:

We are given a Queue data structure that supports standard operations like Enqueuer () and Dequeuer (). We need to implement a Stack data structure using only instances of Queue and queue operations allowed on the instances.

EX NO: 2.1

Date:

# ENQUEUE () AND DEQUEUE ()

## AIM:

To write a C-Program to implement the stack data structure using only instances of queue and queue operations.

## PESUDOCODE:

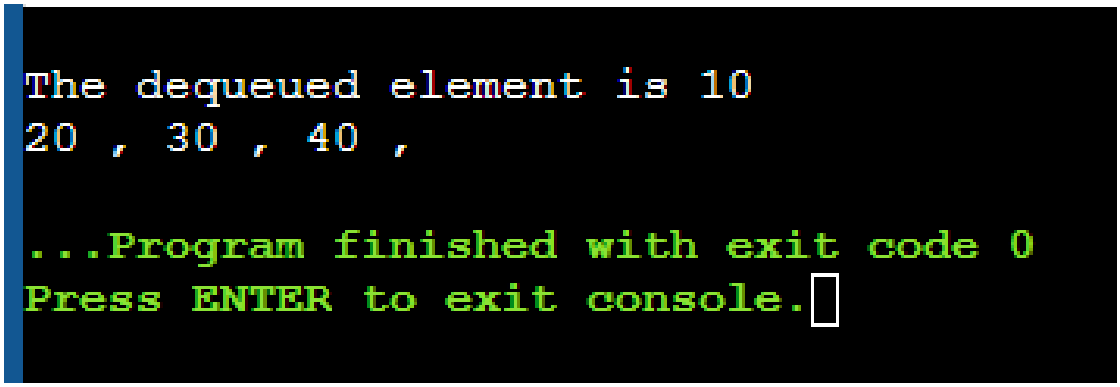
```
Declare stack1, stack2,top1,top2 and count
if(top1==N-1)
it will print stack overflow.
else
top1++;
if(top1==-1)
it will print stack is empty
else
top1--;
if(top2==N-1)
it will print stack is fill
else
top2++;
BEGIN
if((top1==-1) && (top2==-1))
Then it will print queue is empty.
else
for(int i=0;i<count;i++)
int element = pop1();
push2(element);
int b= pop2();
Then, print the dequeue elements
Display,
enqueue(10);
enqueue(20);
enqueue(30);
dequeue();
enqueue(40);
END
```

**PROGRAM:**

```
#include<stdio.h>
#define N 5
int stack1[5], stack2[5];
int top1=-1, top2=-1;
int count=0;
void push1(int data)
{
if(top1==N-1)
{
printf("\n Stack is overflow...");
}
else
{
top1++;
stack1[top1]=data;
}
}
int pop1()
{
if(top1==-1)
{
printf("\nStack is empty..");
}
else
{
int a=stack1[top1];
top1--;
return a;
}
}
void push2(int x)
{
if(top2==N-1)
{
printf("\nStack is full..");
}
else
{
top2++;
stack2[top2]=x;
}
}
int pop2()
{
int element = stack2[top2];
top2--;
return element;
```

```
}
void enqueue(int x)
{
push1(x);
count++;
}
void dequeue()
{
if((top1==-1) && (top2==-1))
{
printf("\nQueue is empty");
}
else
{
for(int i=0;i<count;i++)
{
int element = pop1();
push2(element);
}
int b= pop2();
printf("\nThe dequeued element is %d", b);
printf("\n");
count--;
for(int i=0;i<count;i++)
{
int a = pop2();
push1(a);
}
}}
void display()
{
for(int i=0;i<=top1;i++)
{
printf("%d , ", stack1[i]);
}
}
void main()
{
enqueue(10);
enqueue(20);
enqueue(30);
dequeue();
enqueue(40);
display();
}
```

**OUTPUT:**




**RESULT:**

Thus the program to implement the stack data structure using only instances of queue and queue operations.

## QUESTION 2:

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

**Example:**

Given [0,1,0,2,1,0,1,3,2,1,2,1], return 6.


The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1].

In this case,6 units of rain water (blue section) are being trapped.

| EX NO: 2.1<br>Date: | **RAIN WATER TRAPPING** |
|---|---|

## AIM:

To write a C-Program to determine the amount of water which is able to trap after the raining.

## PESUDOCODE:

Declare max(int x,int y),left, right and water

Return the condition (x>y)?x:y;

Declare maxLeft=heights [left] and maxRight=heights [right]

BEGIN

If the condition heights[left]<=heights[right],then

maxLeft = max(maxLeft, heights[left]);

water += (maxLeft - heights[left]);

Otherwise,it will execute

maxRight = max(maxRight, heights[right]);

water += (maxRight - heights[right]);

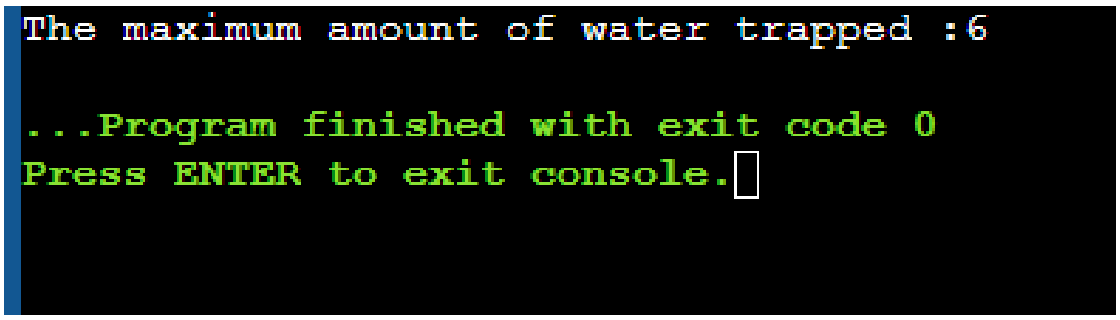
END

## PROGRAM:

```
#include <stdio.h>
int max(int x, int y) {
return (x > y) ? x : y;
}
int trap(int heights[], int n)
{
int left = 0, right = n - 1, water = 0;
int maxLeft = heights[left];
int maxRight = heights[right];
while (left < right)
{
if (heights[left] <= heights[right])
{
left++;
```

```
maxLeft = max(maxLeft, heights[left]);
water += (maxLeft - heights[left]);
}
else {
right--;
maxRight = max(maxRight, heights[right]);
water += (maxRight - heights[right]);
}
}
return water;
}
int main(void)
{
int heights[] = { 0,1,0,2,1,0,1,3,2,1,2,1};
int n = sizeof(heights) / sizeof(heights[0]);
printf("The maximum amount of water trapped :%d",
trap(heights, n));
return 0;
}
```

**OUTPUT:**




**RESULT:**

Thus the program to determine the amount of water which is able to trap after raining.