**Q. WAP to implement a stack using two queues such that the push operation runs in constant time and the pop operation runs in linear time.**

#include<stdio.h>

#include<stdlib.h>

int queue\_1[10], queue\_2[10], num, rear = -1, front = 0, size;

void push()

{

if (rear == size)

{

printf("OVERFLOW");

}

else

{

printf("Enter new element : ");

scanf("%d", & num);

rear = rear + 1;

queue\_1[rear] = num;

}

}

void swap\_1()

{

int i, v[10];

for (i = front; i < rear; i++)

{

v[i] = queue\_1[i];

queue\_1[i] = queue\_2[i];

queue\_2[i] = v[i];

}

}

void swap\_2()

{

int i, x[10];

for (i = front; i < rear; i++)

{

x[i] = queue\_1[i];

queue\_1[i] = queue\_2[i];

queue\_2[i] = x[i];

}

}

void pop()

{

if (rear == -1)

{

printf("UNDERFLOW");

}

else

{

int i;

for (i = front; i < rear; i++)

{

swap\_1();

swap\_2();

rear = rear - 1;

}

}

}

void display()

{

int i;

printf("Your stack contains : \n");

for (i = front; i <= rear; i++)

{

printf("Stack[%d] : %d \n", i, queue\_1[i]);

}

}

void main()

{

int choice, j;

printf("\nEnter the size of the stack : ");

scanf("%d", & size);

for (j = 1; j <= size; j++)

{

printf("Enter element %d : ",j);

scanf("%d", & num);

rear = rear + 1;

queue\_1[rear] = num;

}

display();

while (1)

{

printf("\nMENU : 1-push, 2-pop, 3-exit\n");

printf("Enter Choice : ");

scanf("%d", & choice);

switch (choice)

{

case 1:

{

push();

display();

break;

}

case 2:

{

pop();

display();

break;

}

case 3:

{

exit(1);

break;

}

default:

{

printf("\nError! Enter another no. : ");

}

}

}

}

#include <stdio.h>

#include <stdlib.h>

#define QUEUE\_EMPTY\_MAGIC 0xdeadbeef

typedef struct queue

{

int \*arr;

int rear, front, count, max;

} queue;

queue \*queue\_allocate(int n);

void queue\_insert(queue \* q, int v);

int queue\_remove(queue \* q);

int queue\_count(queue \* q);

int queue\_is\_empty(queue \* q);

void stack\_push(queue \* q, int v)

{

queue\_insert(q, v);

}

int stack\_pop(queue \* q) {

int i, n = queue\_count(q);

int removed\_element;

for (i = 0; i < (n - 1); i++) {

removed\_element = queue\_remove(q);

queue\_insert(q, removed\_element);

}

removed\_element = queue\_remove(q);

return removed\_element;

}

int stack\_is\_empty(queue \* q) {

return queue\_is\_empty(q);

}

int stack\_count(queue \* q) {

return queue\_count(q);

}

int queue\_count(queue \* q) {

return q->count;

}

queue \*

queue\_allocate(int n) {

queue \*queue;

queue = malloc(sizeof(queue));

if (queue == NULL)

return NULL;

queue->max = n;

queue->arr = malloc(sizeof(int) \* n);

queue->rear = n - 1;

queue->front = n - 1;

return queue;

}

void queue\_insert(queue \* q, int v) {

if (q->count == q->max)

return;

q->rear = (q->rear + 1) % q->max;

q->arr[q->rear] = v;

q->count++;

}

int queue\_remove(queue \* q) {

int retval;

/\* magic number if queue is empty \*/

if (q->count == 0)

return QUEUE\_EMPTY\_MAGIC;

q->front = (q->front + 1) % q->max;

retval = q->arr[q->front];

q->count--;

return retval;

}

int queue\_is\_empty(queue \* q) {

return (q->count == 0);

}

void queue\_display(queue \* q) {

int i = (q->front + 1) % q->max, elements = queue\_count(q);

while (elements--) {

printf("[%d], ", q->arr[i]);

i = (i >= q->max) ? 0 : (i + 1);

}

}

#define MAX 128

int main(void) {

queue \*q;

int x, select;

q = queue\_allocate(MAX);

do {

printf("\n[1] Push\n[2] Pop\n[0] Exit");

printf("\nChoice: ");

scanf(" %d", &select);

switch (select) {

case 1:

printf("\nEnter value to Push:");

scanf(" %d", &x);

/\* Pushing \*/

stack\_push(q, x);

printf("\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nCurrent Queue:\n");

queue\_display(q);

printf("\n\nPushed Value: %d", x);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

break;

case 2:

/\* Popping \*/

x = stack\_pop(q);

printf("\n\n\n\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nCurrent Queue:\n");

queue\_display(q);

if (x == QUEUE\_EMPTY\_MAGIC)

printf("\n\nNo values removed");

else

printf("\n\nPopped Value: %d", x);

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

break;

case 0:

printf("\nQutting.\n");

return 0;

default:

printf("\nQutting.\n");

return 0;

}

} while (1);

return 0;

}

**OUTPUT :**

