Assignment

**Question 1:** Write a source code how to implement efficiently multiple queues in a single array.

1. Multiple queues are 4

2. The Queue, name Q1 should contain 5 elements

3. Second Queue, name Q2 should contain 3 elements

4. Third Queue, name Q3 should contain 2 elements

5. Fourth Queue, name Q4 should contain 6 elements

6. Delete operation should perform on the maximum number of elements in queue

7. Insert operation should perform on the minimum number of elements in the Queue

8. Perform the sequence of operations: Delete, Delete, Delete, Insert, Insert, Delete, Display all four queues.

**Source\_Code:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#define queue\_count 4

#define max\_queue\_length 10

int array[queue\_count \* max\_queue\_length];

int front[queue\_count],rear[queue\_count];

void question\_elements(){

rear[0] = 5;

rear[1] = 13;

rear[2] = 22;

rear[3] = 36;

for(int i=0;i<5;i++){

array[i] = i+1; //insering elements in array - 1st que

}

for(int i=10;i<13;i++){

array[i] = i+1; //insering elements in array - 2nd que

}

for(int i=20;i<22;i++){

array[i] = i+1; //insering elements in array - 3rd que

}

for(int i=30;i<36;i++){

array[i] = i+1; //insering elements in array - 4th que

}

}

void display(){

for(int qno=0; qno<queue\_count; qno++){

printf("Queue %d elements: ",qno);

int start = front[qno], end = rear[qno];

while(start != end){

printf("%d \t",array[start]);

start++;

}

printf("\n");

}

}

//Finding max number of elements que and returning that que number

int max\_elements\_q(){

int max = -1;

int qno = 0;

for(int i = 0;i<queue\_count;i++){

if(rear[i] - front[i] > max){

max = rear[i] - front[i];

qno = i;

}

}

return qno;

}

//Finding min number of elements que and returning that que number

int min\_elements\_q(){

int min = max\_queue\_length+1;

int qno = 0;

for(int i = 0;i<queue\_count;i++){

if(rear[i] - front[i] < min){

min = rear[i] - front[i];

qno = i;

}

}

return qno;

}

//inserting elements to que

int insert(int value){

int qno = min\_elements\_q();

if(rear[qno] >= (qno+1) \* max\_queue\_length){

printf("Queue %d overflow", qno);

display();

return -1;

}

array[rear[qno]] = value;

rear[qno]++;

printf("\n Queues after insertion \n");

display();

return 1;

}

int delete(){

int qno = (int)max\_elements\_q();

if(front[qno] == rear[qno]){

printf("Queue %d underflow", qno);

display();

return -1;

}

array[front[qno]] = -1;

front[qno]++;

printf("\n Queues after deletion \n");

display();

return 1;

}

int main() {

int i;

for(i = 0; i<queue\_count; i++)

front[i] = rear[i] = i\*max\_queue\_length;

question\_elements();

display();

delete();

delete();

delete();

insert(100);

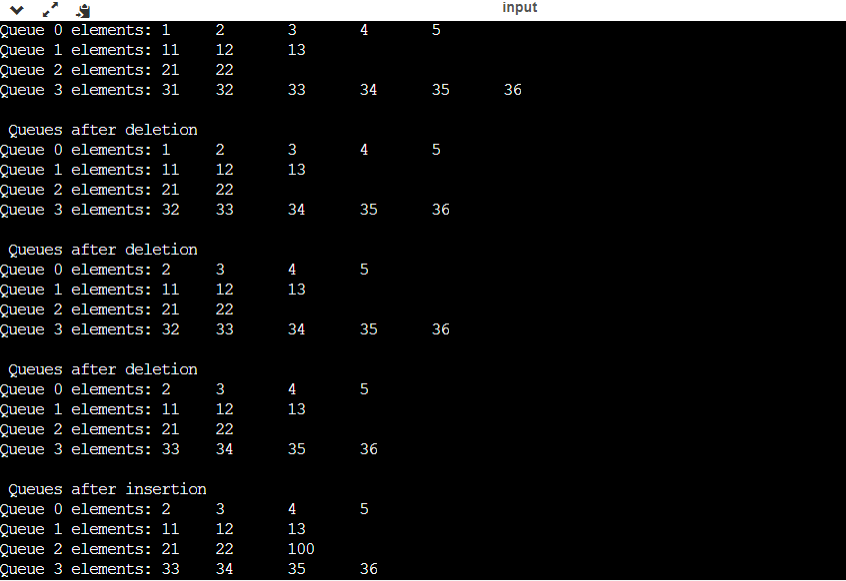
insert(102);

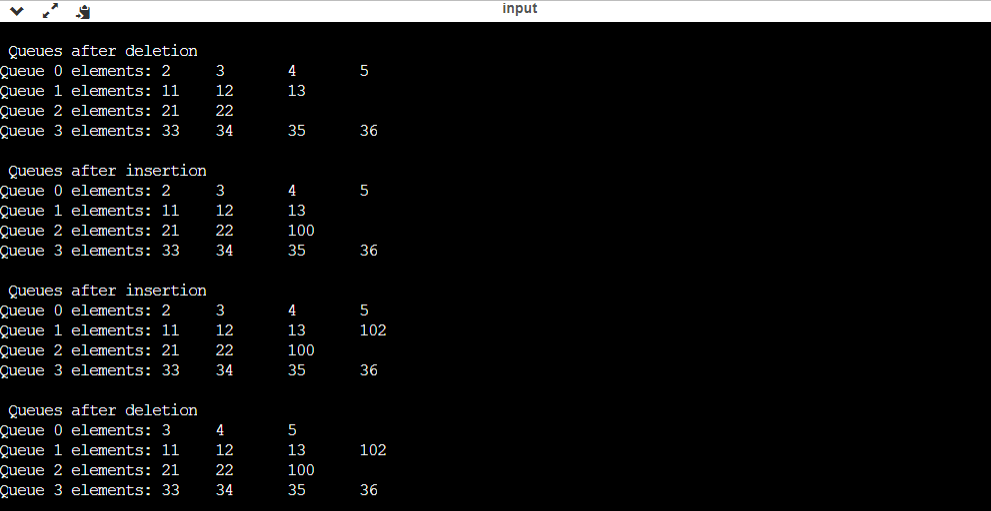
delete();

return 0;

}

**Output:**

****

****

**Question 2:**

Split a Circular Linked List into two halves Given a Circular Linked List of size N, split it into two halves circular lists. If there are odd number of nodes in the given circular linked list then out of the resulting two halved lists, first list should have one node more than the second list. The resultant lists should also be circular lists and not linear lists.

**Source\_Code**:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

struct Node {

int data;

struct Node \*next;

};

struct Node \*head = NULL;

void display()

{

struct Node \*ptr;

ptr =head;

while(ptr->next != head)

{

printf("%d -->",ptr->data);

ptr = ptr->next;

}

printf("%d ",ptr->data);

printf("\n");

}

void lastinsert()

{

struct Node \*newNode,\*temp;

int ele;

newNode = (struct Node \*)malloc(sizeof(struct Node));

printf("\n Enter data");

scanf("%d",&ele);

newNode->data = ele;

if(newNode == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

if(head == NULL)

{

head = newNode;

newNode -> next = head;

} //end of if

else

{

temp = head;

while(temp -> next != head)

{

temp = temp -> next;

}

temp -> next = newNode;

newNode -> next = head;

}//end of else

printf("\n Node inserted \n");

}

}

int main()

{

struct Node \*tmp1,\*tmp2, \*ptr1,\*ptr2,\*head2;

int cnt,break\_point;

printf("\n Enter number of nodes ");

scanf("%d",&cnt);

for(int i=0;i<cnt;i++)

{

lastinsert();

}

printf("\n Displaying the nodes \n");

display();

if(cnt % 2 == 0)

{

break\_point = (cnt/2) ;

}

else

{

break\_point = (cnt/2)+1;

}

tmp1= head;

for(int j=0;j<break\_point-1;j++)

{

tmp1 = tmp1->next;

}

head2 = tmp1->next;

tmp1->next = head;

tmp2 = head2;

//display 1st list

printf("List1 is \n");

ptr1 = head;

while(ptr1->next != head)

{

printf("%d -->",ptr1->data);

ptr1 = ptr1->next;

}

printf("%d ",ptr1->data);

//After the break\_point

while(tmp2->next!= head)

{

tmp2 = tmp2->next;

}

tmp2->next = head2;

ptr2 = head2;

printf("\n");

//display 2nd list

printf("List2 is \n");

while(ptr2->next != head2)

{

printf("%d -->",ptr2->data);

ptr2 = ptr2->next;

}

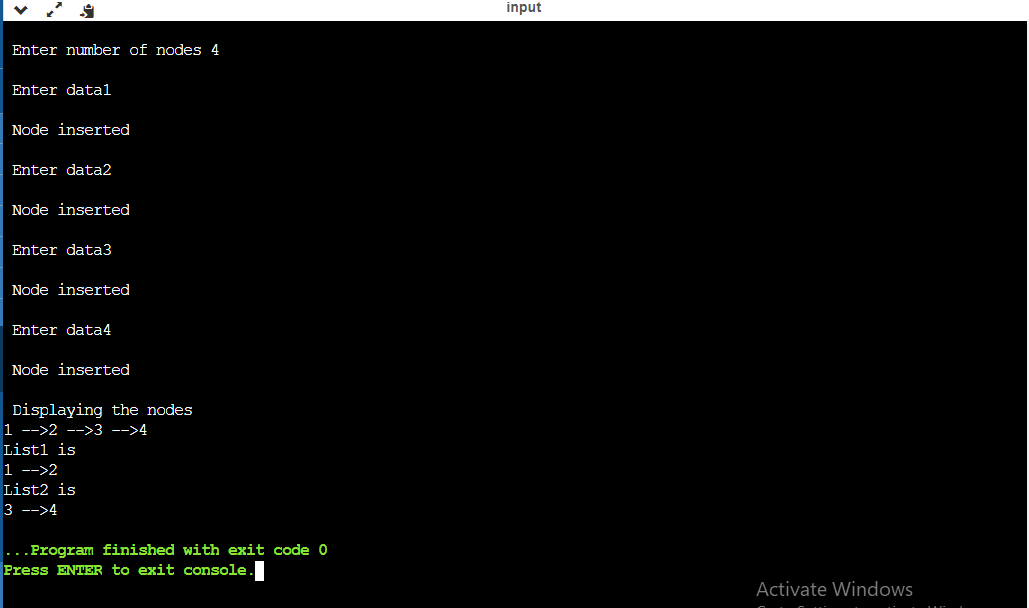
printf("%d",ptr2->data);

return 0;

}

**Output:**





**Question 3:**

Write a source code on how to implement a priority queue Consider,

1. Create Queue using arrays

2. Input for the Queue is the name of the student, Gate score of the student(G1)

3. Perform the operations:

Insert(name,G1),

Insert(name,G2(G2>G1)),

Insert(name,G3(G3<G2)),

Insert(name,G4(G4=G2)),

Delete,

Insert(name,G5(G5<G2))

Delete,

Display the name of the student along with Gate Score

**Source\_Code:**

#include <stdio.h>

#include<string.h>

#include<conio.h>

#include<stdlib.h>

// Structure for the elements in the

// priority queue

struct student {

int g\_score;

int priority;

char name[30];

};

// Store the element of a priority queue

struct student pr\_que[1000];

// Pointer to the last index

int size = -1;

// Function to insert a new record into priority queue

void Enque(int g\_score, int priority, char name[])

{

// Increase the size

size++;

// Insert the element

strcpy(pr\_que[size].name, name);

pr\_que[size].g\_score = g\_score;

pr\_que[size].priority = priority;

}

// Function to check the top element

int Check\_Top()

{

int high\_Priority = -1;

int index = -1;

// Check for the element with

// highest priority

for (int i = 0; i <= size; i++) {

if (high\_Priority == pr\_que[i].g\_score && index > -1 &&

pr\_que[index].name < pr\_que[i].name)

{

high\_Priority = pr\_que[i].g\_score;

index = i;

}

else if (high\_Priority < pr\_que[i].g\_score)

{

high\_Priority = pr\_que[i].g\_score;

index = i;

}

}

// return the priority index

return index;

}

/\* Below function removes element with high priority\*/

void Deque()

{

// We find the index of element with highest priority

int ind = Check\_Top();

// Shifting the records such that all the records next to

//priority index will start at priority index

for (int i = ind; i < size; i++) {

pr\_que[i] = pr\_que[i + 1];

}

// Decrease the size of priority after deletion

size--;

}

int main()

{

Enque(590, 5,"Ravi");

Enque(650, 7, "Sai");

Enque(610, 6,"Krish");

Enque(650, 7,"Teja");

printf("\n\n After Enqueue...");

for(int j=0; j<=size;j++)

{

printf("\n Name is %s Gate score is %d ",pr\_que[j].name,pr\_que[j].g\_score);

}

// Dequeue the top element

Deque();

printf("\n\n After Dequeue...");

for(int j=0; j<=size;j++)

{

printf("\n Name is %s Gate score is %d ",pr\_que[j].name,pr\_que[j].g\_score);

}

printf("\n\n Another record is inserted \n");

Enque(410,4,"Ram");

// Dequeue the top element

Deque();

printf("\n\n After Dequeue...");

for(int j=0; j<=size;j++)

{

printf("\n Name is %s Gate score is %d ",pr\_que[j].name,pr\_que[j].g\_score);

}

printf("\n\n Finally in Queue...");

for(int j=0; j<=size;j++)

{

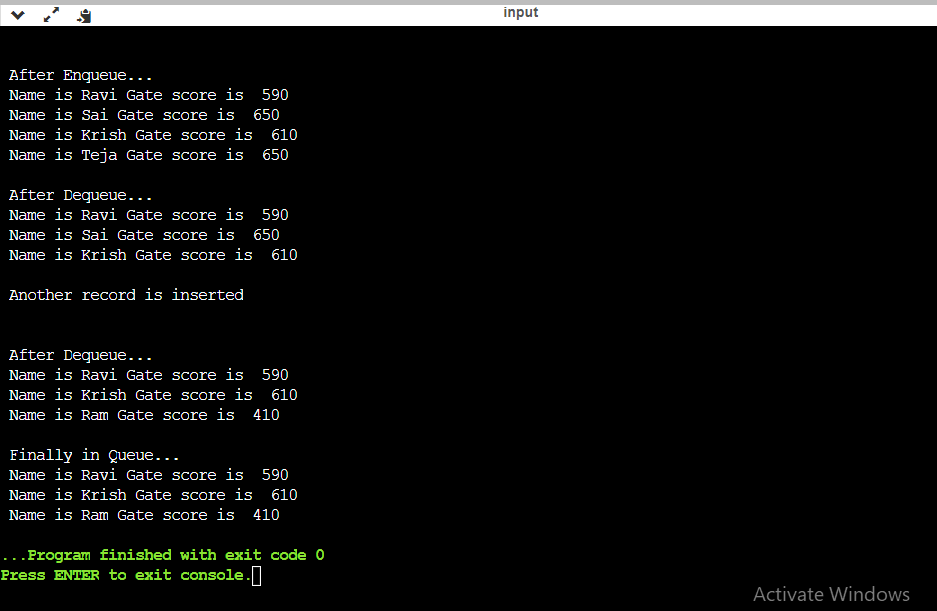
printf("\n Name is %s Gate score is %d ",pr\_que[j].name,pr\_que[j].g\_score);

}

return 0;

}

**Output:**

****