**Lab - Assignement-4**

**Name: kanjariya Hemang**

**Roll No: 3**

**Class: FY-B**

**Q1- Write a program to circular Queue With array Implementation.**

/\*Name:Kanjariya hemang

Roll No: 3

Class : FY- B\*/

#include <stdio.h>

# define max 6

int queue[max]; // array declaration

int front=-1;

int rear=-1;

// function to insert an element in a circular queue

void enqueue(int element)

{

if(front==-1 && rear==-1) // condition to check queue is empty

{

front=0;

rear=0;

queue[rear]=element;

}

else if((rear+1)%max==front) // condition to check queue is full

{

printf("Queue is overflow..");

}

else

{

rear=(rear+1)%max; // rear is incremented

queue[rear]=element; // assigning a value to the queue at the rear position.

}

}

// function to delete the element from the queue

int dequeue()

{

if((front==-1) && (rear==-1)) // condition to check queue is empty

{

printf("\nQueue is underflow..");

}

else if(front==rear)

{

printf("\nThe dequeued element is %d", queue[front]);

front=-1;

rear=-1;

}

else

{

printf("\nThe dequeued element is %d", queue[front]);

front=(front+1)%max;

}

}

// function to display the elements of a queue

void display()

{

int i=front;

if(front==-1 && rear==-1)

{

printf("\n Queue is empty..");

}

else

{

printf("\nElements in a Queue are :");

while(i<=rear)

{

printf("%d,", queue[i]);

i=(i+1)%max;

}

}

}

int main()

{

int choice=1,x; // variables declaration

printf("Name:Kanjariya Hemang\nRoll No:3\nClass:FY-B\n");

while(choice<4 && choice!=0) // while loop

{

printf("\n Press 1: Insert an element");

printf("\nPress 2: Delete an element");

printf("\nPress 3: Display the element");

printf("\n Press 4: Exit\n") ;

printf("\nEnter your choice : ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the element which is to be inserted:");

scanf("%d", &x);

enqueue(x);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

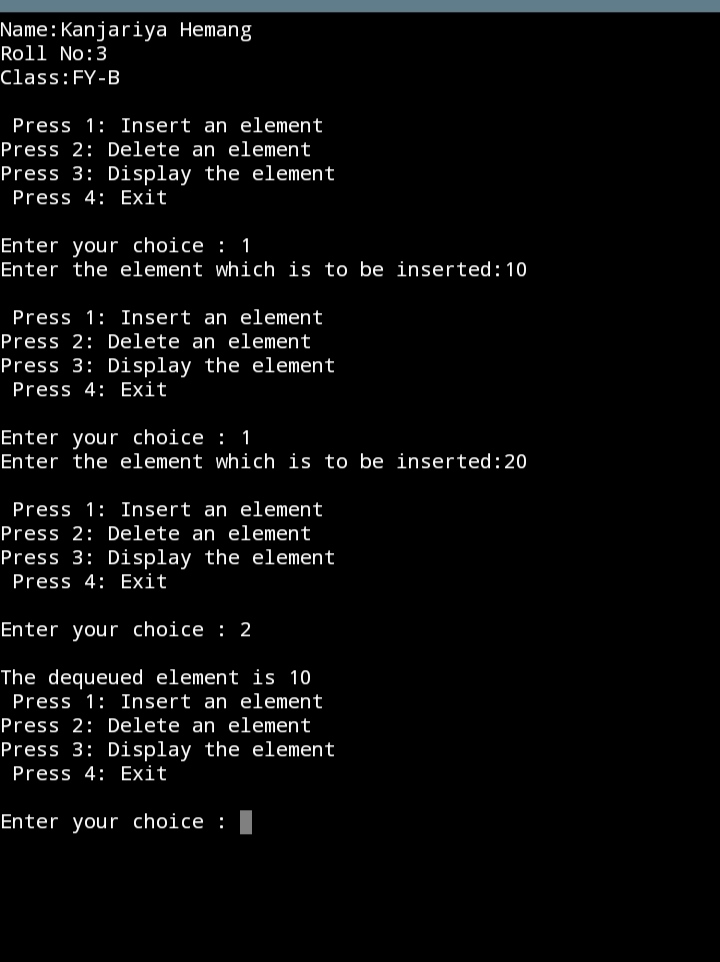
break;

}}

return 0;

}

**Output:-**



**Q2- Write a program for deque and perform all operation.**

#include <stdio.h>

#define SIZE 5

int items[SIZE];

int front = -1, rear = -1;

// Check if the queue is full

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;

return 0;

}

// Check if the queue is empty

int isEmpty() {

if (front == -1) return 1;

return 0;

}

// Adding an element

void enQueue(int element) {

if (isFull())

printf("\n Queue is full!! \n");

else {

if (front == -1) front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n Inserted -> %d", element);

}

}

// Removing an element

int deQueue() {

int element;

if (isEmpty()) {

printf("\n Queue is empty !! \n");

return (-1);

} else {

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

}

// Q has only one element, so we reset the

// queue after dequeing it. ?

else {

front = (front + 1) % SIZE;

}

printf("\n Deleted element -> %d \n", element);

return (element);

}

}

// Display the queue

void display() {

int i;

if (isEmpty())

printf(" \n Empty Queue\n");

else {

printf("\n Front -> %d ", front);

printf("\n Items -> ");

for (i = front; i != rear; i = (i + 1) % SIZE) {

printf("%d ", items[i]);

}

printf("%d ", items[i]);

printf("\n Rear -> %d \n", rear);

}

}

int main() {

// Fails because front = -1

deQueue();

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

// Fails to enqueue because front == 0 && rear == SIZE - 1

enQueue(6);

display();

deQueue();

display();

enQueue(7);

display();

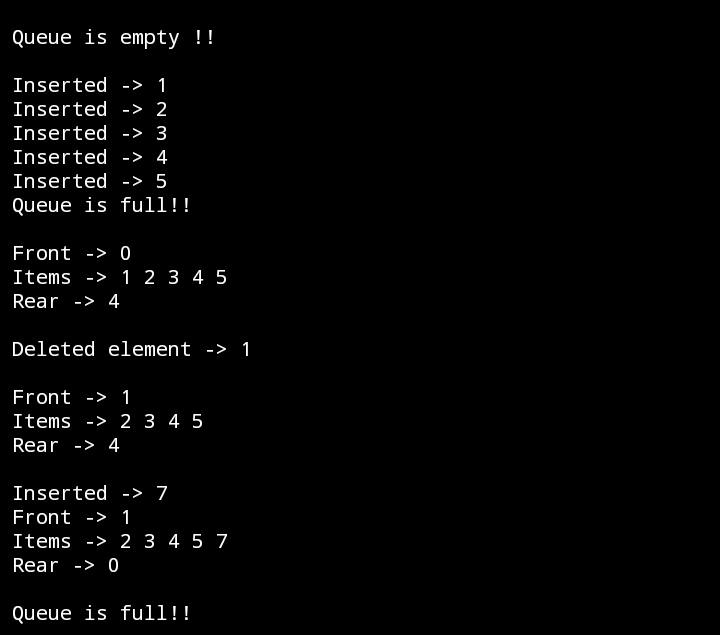
// Fails to enqueue because front == rear + 1

enQueue(8);

return 0;

}

**Output:-**



**Q3- Write a Program for Priority Queue.**

**#include <stdio.h>**

**int size = 0;**

**void swap(int \*a, int \*b) {**

**int temp = \*b;**

**\*b = \*a;**

**\*a = temp;**

**}**

**// Function to heapify the tree**

**void heapify(int array[], int size, int i) {**

**if (size == 1) {**

**printf("Single element in the heap");**

**} else {**

**// Find the largest among root, left child and right child**

**int largest = i;**

**int l = 2 \* i + 1;**

**int r = 2 \* i + 2;**

**if (l < size && array[l] > array[largest])**

**largest = l;**

**if (r < size && array[r] > array[largest])**

**largest = r;**

**// Swap and continue heapifying if root is not largest**

**if (largest != i) {**

**swap(&array[i], &array[largest]);**

**heapify(array, size, largest);**

**}**

**}**

**}**

**// Function to insert an element into the tree**

**void insert(int array[], int newNum) {**

**if (size == 0) {**

**array[0] = newNum;**

**size += 1;**

**} else {**

**array[size] = newNum;**

**size += 1;**

**for (int i = size / 2 - 1; i >= 0; i--) {**

**heapify(array, size, i);**

**}**

**}**

**}**

**// Function to delete an element from the tree**

**void deleteRoot(int array[], int num) {**

**int i;**

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

**if (num == array[i])**

**break;**

**}**

**swap(&array[i], &array[size - 1]);**

**size -= 1;**

**for (int i = size / 2 - 1; i >= 0; i--) {**

**heapify(array, size, i);**

**}**

**}**

**// Print the array**

**void printArray(int array[], int size) {**

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

**printf("%d ", array[i]);**

**printf("\n");**

**}**

**// Driver code**

**int main() {**

**int array[10];**

**insert(array, 3);**

**insert(array, 4);**

**insert(array, 9);**

**insert(array, 5);**

**insert(array, 2);**

**printf("Max-Heap array: ");**

**printArray(array, size);**

**deleteRoot(array, 4);**

**printf("After deleting an element: ");**

**printArray(array, size);**

**}**

**Output:-**

s