Experiment 2.4

|  |  |
| --- | --- |
| **NAME- Vivek**  **BRANCH-** CSE **SEMESTER-** 3rd  **SUBJECT-** DATA STRUCTURES | **UID-**  **CLASS AND GROUP-**  **DATE OF PERFORMANCE-** 03/09/2022  **SUBJECT CODE-** 21CSH-211 |

# AIM OF THE EXPERIMENT:

Write a program to demonstrate the implementation of various operations on a linear queue and circular represented using a linear array.

# ALGORITHM:

## Working of Queue

1. Queue operations work as follows:
2. two pointers FRONT and REAR
3. FRONT track the first element of the queue
4. REAR track the last element of the queue
5. initially, set value of FRONT and REAR to -1

## enQueue Operation

1. check if the queue is full
2. for the first element, set the value of FRONT to 0
3. increase the REAR index by 1
4. add the new element in the position pointed to by REAR

## deQueue Operation

1. check if the queue is empty
2. return the value pointed by FRONT
3. increase the FRONT index by 1
4. for the last element, reset the values of FRONT and REAR to -1

# PROGRAM CODE:

## Linear Queue:-

#include <iostream> using namespace std;

struct Queue {

int front, rear, capacity;

int\* queue; Queue(int c){

front = rear = 0; capacity = c; queue = new int;

}

~Queue() { delete[] queue; } void Enqueue(int data){

if (capacity == rear) { printf("\nQueue is full\n"); return;

}

else {

queue[rear] = data; rear++;

}

return;

}

void Dequeue(){

if (front == rear) { printf("\nQueue is empty\n"); return;

}

else {

for (int i = 0; i < rear - 1; i++) { queue[i] = queue[i + 1];

}

rear--;

}

return;

}

void Display(){ int i;

if (front == rear) { printf("\nQueue is Empty\n"); return;

}

for (i = front; i < rear; i++) { printf(" %d <-- ", queue[i]);

}

return;

}

void Front(){

if (front == rear) { printf("\nQueue is Empty\n"); return;

}

printf("\nFront Element is: %d", queue[front]); return;

}

};

int main(void){ Queue q(4); q.Display(); q.Enqueue(10); q.Enqueue(30); q.Enqueue(50); q.Enqueue(70); q.Display();

q.Enqueue(60); q.Display();

q.Dequeue();

q.Dequeue();

printf("\n\nafter two node deletion\n\n"); q.Display();

q.Front(); return 0;

}

**Circular Queue:-** #include<bits/stdc++.h> using namespace std;

class Queue{

int rear, front,size,\*arr; public:

Queue(int s){

front = rear = -1; size = s;

arr = new int[s];

}

void enQueue(int value); int deQueue();

void displayQueue();

};

void Queue::enQueue(int value){

if ((front == 0 && rear == size-1) || (rear == (front-1)%(size-1))){ printf("\nQueue is Full");

return;

}

else if (front == -1){ front = rear = 0; arr[rear] = value;

}

else if (rear == size-1 && front != 0){ rear = 0;

arr[rear] = value;

}

else{

rear++;

arr[rear] = value;

}

}

int Queue::deQueue(){ if (front == -1){

printf("\nQueue is Empty"); return INT\_MIN;

}

int data = arr[front]; arr[front] = -1;

if (front == rear){ front = -1;

rear = -1;

}

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

else

front++;

return data;

}

void Queue::displayQueue(){ if (front == -1){

printf("\nQueue is Empty"); return;

}

printf("\nElements in Circular Queue are: "); if (rear >= front){

for (int i = front; i <= rear; i++) printf("%d ",arr[i]);

}

else{

for (int i = front; i < size; i++) printf("%d ", arr[i]);

for (int i = 0; i <= rear; i++) printf("%d ", arr[i]);

}

}

int main(){

Queue q(5); q.enQueue(13); q.enQueue(24);

q.enQueue(17); q.enQueue(-4); q.displayQueue();

printf("\nDeleted value = %d", q.deQueue()); printf("\nDeleted value = %d", q.deQueue()); q.displayQueue();

q.enQueue(71); q.enQueue(81); q.enQueue(1); q.displayQueue(); q.enQueue(20); return 0;

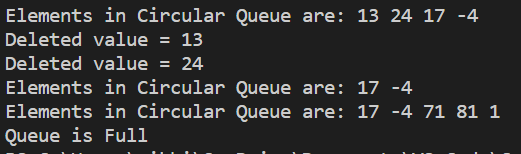
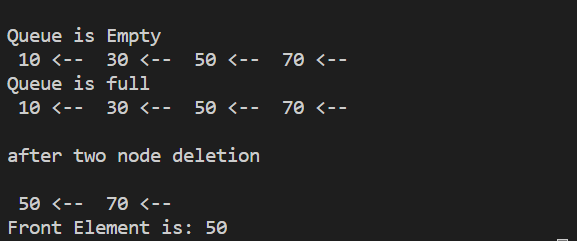
}

# PRORGAM’S EXPLANATION:

Program that performs operations like Front, Rear, enQueue and deQueue on linear and Circular Queue using linear array.

# OUTPUT:

Linear Queue: Circular Queue:



# LEARNING OUTCOMES (What I have learnt):

1. I have learnt about Queue.
2. I have learnt how to perform different operations on Queue.
3. I have learnt about Circular Queue.