|  |  |
| --- | --- |
| https://lh6.googleusercontent.com/NfxikauFutj0NGA6tZLo-9OYuuo3IKL8zEJJ1vcz211S75EzE4cV8OiG_9xiNDI7UXEpwJU3L0HucAlThGrGrN0dBZNqE1lpoGJHT3oGyRmOalINq0YRgVxjjMgLbZ3n8Pq0wpa0thnjQcjh_kKTHUlnU-ExSfeM05HoXY1-s1p-Qtg0OyqjFjrwB05Nj4mxriktv4A | SVKM’s NMIMS  School of Technology Management & Engineering Navi Mumbai Campus |
| Department of Computer Engineering |

|  |  |
| --- | --- |
| Name: Yash Patil | SAP ID: 70472200140 |
| Semester: III | Year: II |
| Subject: Data Structures and Algorithm | Roll No.: A176 |
| Practical: 4 | Date: 17/08/2023 |
| Batch: 1 |  |

**Aim:–**

Implementation of simple queues and circular queue using arrays and apply them in real life computer applications - Railway reservation system

**Theory:–**

### What is a Circular Queue?

A circular queue is similar to a linear queue as it is also based on the FIFO (First In First Out) principle except that the last position is connected to the first position in a circular queue that forms a circle. It is also known as a Ring Buffer.

### Operations on Circular Queue

The following are the operations that can be performed on a circular queue:

* Front: It is used to get the front element from the Queue.
* Rear: It is used to get the rear element from the Queue.
* enQueue(value): This function is used to insert the new value in the Queue. The new element is always inserted from the rear end.
* deQueue(): This function deletes an element from the Queue. The deletion in a Queue always takes place from the front end.

**Code/Implementation –**

Circular Queue:

#include<stdio.h>

#define SIZE 5

int arr[SIZE];int front = -1;int rear = -1;

void enqueue(int e)

{

if(front==(rear+1)%SIZE)

{

printf("\Overflow!!\n");

}

else

{

if(rear==-1)

{

rear=0;front=0;

}

else

{

rear=(rear+1)%SIZE;

}

arr[rear]=e;

}

}

void dequeue()

{

if(front==-1)

{

printf("\nUnderflow!!\n");

}

else

{

printf("\nDeleted element: %d",arr[front]);

if(front==rear)

{

front=-1;rear=-1;

}

else

{

front=(front+1)%SIZE;

}

}

}

void display()

{

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

{

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

}

if(front<=rear)

{

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

{

printf("\t%d",arr[i]);

}

}

else

{

for(int j=front;j<=SIZE-1;j++)

{

printf("\t%d",arr[j]);

}

for(int k=0;k<=rear;k++)

{

printf("\t%d",arr[k]);

}

}

}

int main()

{

int ch;int e;

do

{

printf("\nEnter your choice\n1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter element:\t");

scanf("%d",&e);

enqueue(e);

break;

case 2:

dequeue();break;

case 3:

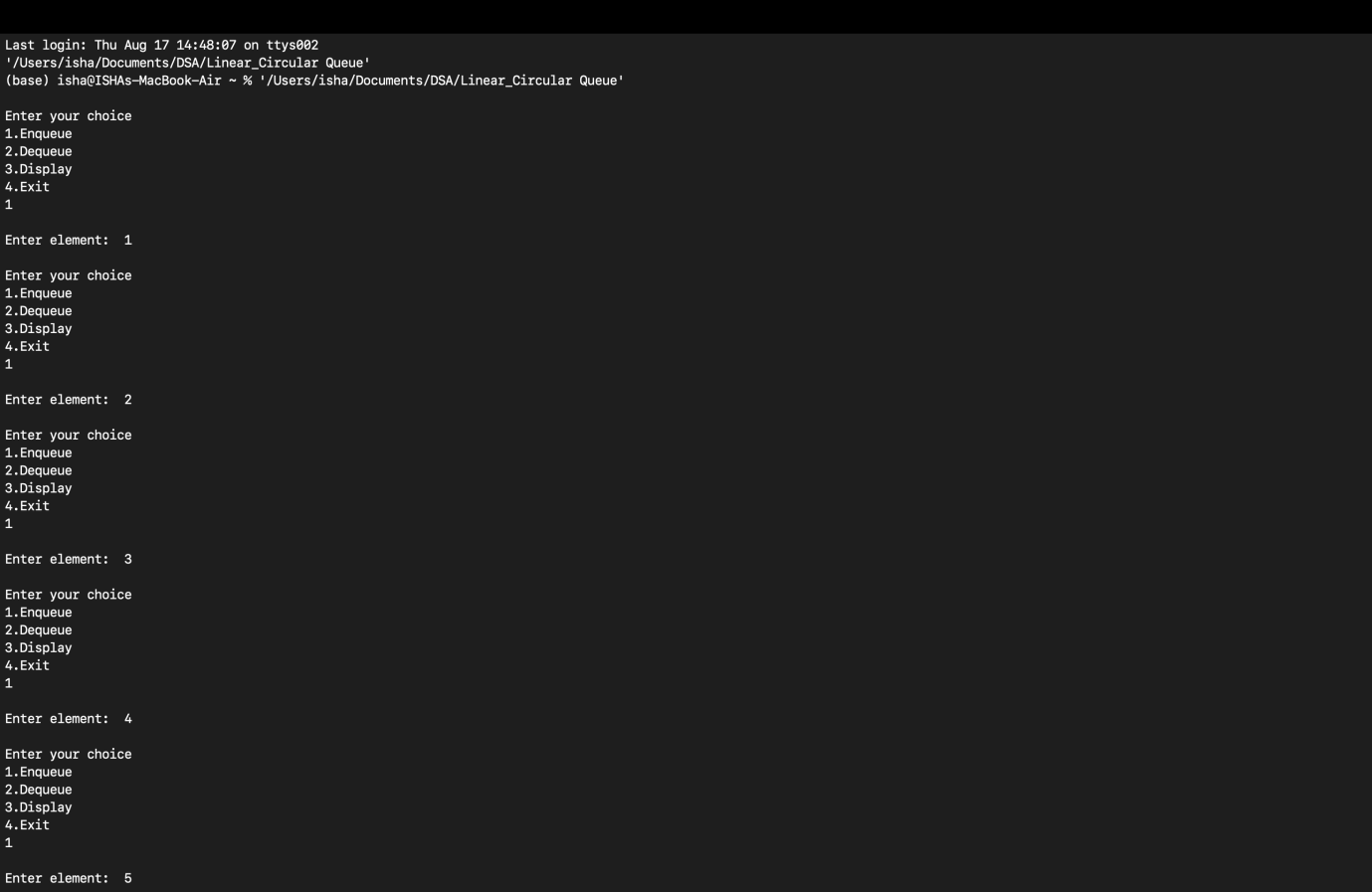
display();break;

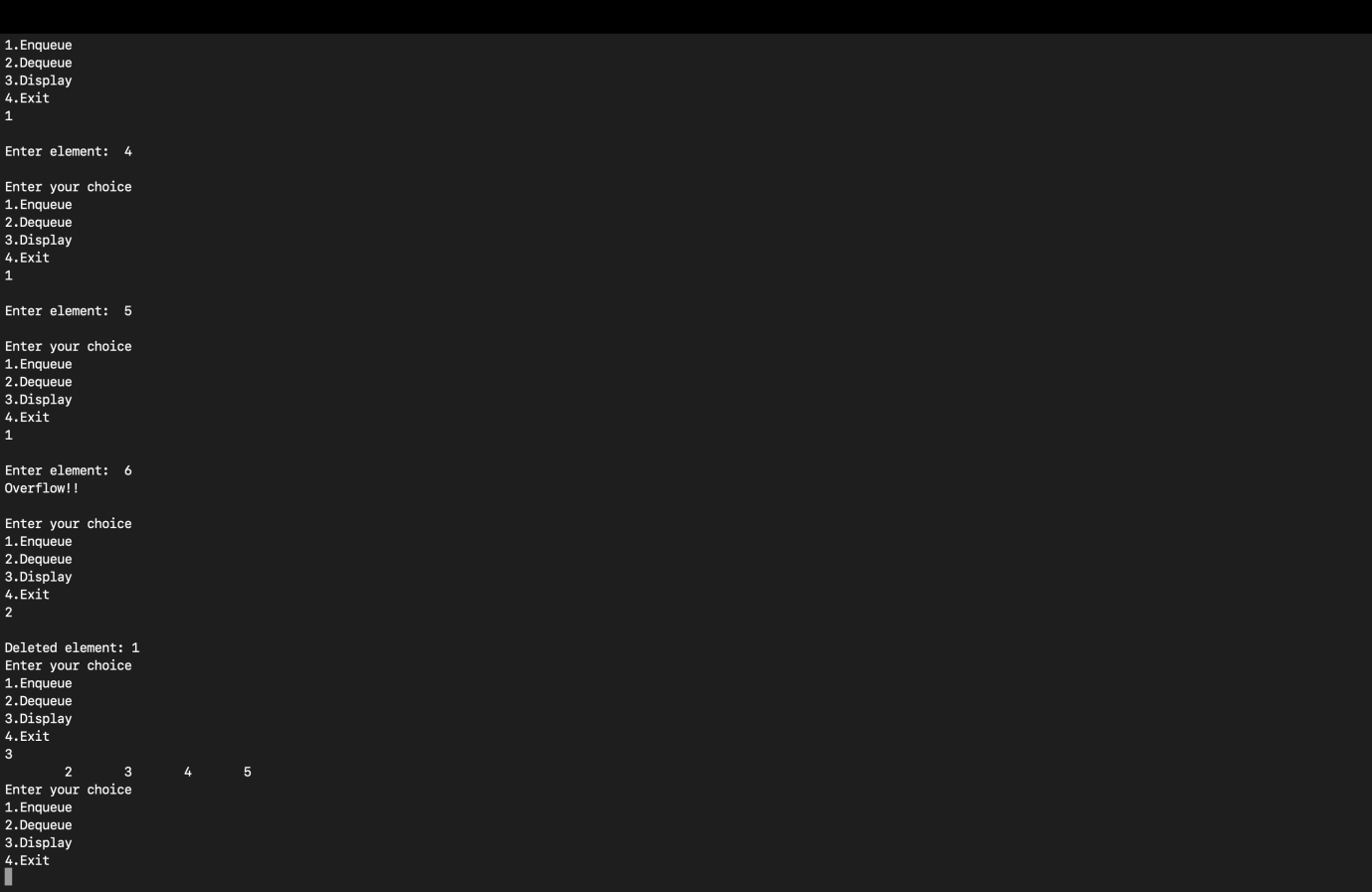
}

}while(ch<4);

}

**Output:-**

****

****

**Conclusion:-**

Was able to study the queue data structure through array.

**Outcome: -** Identified and applied appropriate queue data structure for the given problem