Experiment no 4

NAME: SHREYA SANJAY CHAUDHARI

BRANCH: COMPS DIV: 3 ROLL NO: 04

**Aim:** To implement a Queue using arrays

**Objective:**

1.Understand the Queue Data Structure and its basis operation

2.undertand the method of defining Queue ADT and its basic operation

3.Learn how to create object from an ADT and member function are invoked

**Theory:**

A queue data structure can be implemented using one dimensional array. The queue implemented using array stores only fixed number of data values. The implementation of queue data structure using array is very simple. Just define a one dimensional array of specific size and insert or delete the values into that array by using **FIFO (First in First Out) principle** with the help of variables **'front'** and '**rear**'. Initially both '**front**' and '**rear**' are set to -1. Whenever, we want to insert a new value into the queue, increment '**rear**' value by one and then insert at that position. Whenever we want to delete a value from the queue, then delete the element which is at 'front' position and increment 'front' value by one.

**Queue Operations using Array**

Queue data structure using array can be implemented as follows...  
  
Before we implement actual operations, first follow the below steps to create an empty queue.

* **Step 1 -**Include all the **header files** which are used in the program and define a constant **'SIZE'** with specific value.
* **Step 2 -**Declare all the **user defined functions** which are used in queue implementation.
* **Step 3 -**Create a one dimensional array with above defined SIZE (**int queue[SIZE]**)
* **Step 4 -**Define two integer variables **'front'** and '**rear**' and initialize both with **'-1'**. (**int front = -1, rear = -1**)
* **Step 5 -**Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue.

**enQueue(value) - Inserting value into the queue**

In a queue data structure, enQueue() is a function used to insert a new element into the queue. In a queue, the new element is always inserted at **rear** position. The enQueue() function takes one integer value as a parameter and inserts that value into the queue. We can use the following steps to insert an element into the queue...

* **Step 1 -**Check whether **queue** is **FULL**. (**rear == SIZE-1**)
* **Step 2 -**If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.
* **Step 3 -**If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

**dqQueue - Deleting a value from the Queue**

In a queue data structure, deQueue() is a function used to delete an element from the queue. In a queue, the element is always deleted from **front** position. The deQueue() function does not take any value as parameter. We can use the following steps to delete an element from the queue...

* **Step 1 -**Check whether **queue** is **EMPTY**. (**front == rear**)
* **Step 2 -**If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
* **Step 3 -**If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front** == **rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front** = **rear** = **-1**).

**display() - Displays the elements of a Queue**

We can use the following steps to display the elements of a queue...

* **Step 1 -**Check whether **queue** is **EMPTY**. (**front == rear**)
* **Step 2 -**If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.
* **Step 3 -**If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i** = **front+1**'.
* **Step 4 -**Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value reaches to **rear** (**i** <= **rear**)

**Algorithm:**

START

**Step 2**: Store the element which is to be inserted

**Step 3:** Check if **REAR= MAX-1** then write Queue is full

             else go to step 5

**Step 4**: Check whether **REAR=-1** then set **FRONT=REAR=0**

              else

              set **REAR=REAR+1**

**Step 5**: Set **QUEUE [REAR]=Value**

**Step 6:**STOP

Code:#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#define max 100

void enque();

void deque();

void display();

int queue[max],front=-1,rear=-1;

int main(){

int value,choice;

do{

printf("\n\t \*\*\*\*\*MAIN MENU\*\*\*\*\*");

printf("\n 1.Insertion\n 2.Deletion\n 3.Dislay \n 4.Exit\n");

printf("Enter choice:");

scanf("%d",&choice);

switch(choice){

case 1: enque();

break;

case 2: deque();

break;

case 3: display();

break;

case 4: break;

default: printf("\n INVALID CHOICE!!!");

break;

}

}

while(choice!=5);

return 0;

}

void enque() {

int num;

if(front==0 && rear==max-1){

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

}

else{

printf("\n Enter the data to be inserted:");

scanf("%d",&num);

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

front=rear=0;

queue[rear]=num;

}

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

rear=0;

queue[rear]=num;}

else{

rear++;

queue[rear]=num;

}

}}

void deque(){

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

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

}

else{

printf("\n the deleted element is %d",queue[front]);

if(front==rear){

front=rear=-1;

}

else if(front==max-1){

front=0;

}

else{

front++;

}}}

void display(){

int i;

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

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

}

else

{

printf("\n the elements in the queue are:");

if(front<rear)

{

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

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

}}

else if(front>rear){

for(i=front;i<max;i++){

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

}

for(i=0;i<=rear;i++){

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

}}

else{

printf("%d\t",queue[front]);

}}}

**Output :**



**Conclusion:** In conclusion, a linear queue with the shortest job first (SJF) scheduling algorithm is a valuable combination for managing tasks or processes in a way that optimizes execution time and minimizes waiting times.