Name:Shakib Ahmad Shaikh

S.E Comps:3 Roll:no.37 Batch-B

Experiment No. 4: Implement of Linear Queue ADT using Array

Aim: To implement a Queue using arrays.

Objective:

1 Understand the Queue data structure and its basis operations.

2. Understand the method of defining Queue ADT and its basic operations.

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

Theory:

A Queue is an ordered collection of items from which items may be deleted at one end

(called the front of the queue) and into which items may be inserted at the other end (the rear of the queue). Queues remember things in first-in-first-out (FIFO) order.

The basic operations in a queue are: Enqueue - Adds an item to the end of queue. Dequeue - Removes an item from the front.

A queue is implemented using a one dimensional array. FRONT is an integer value, which contains the array index of the front element of the array.

REAR is an integer value, which contains the array index of the rear element of the array.

When an element is deleted from the queue, the value of front is increased by one. When an element is inserted into the queue, the value of rear is increased by one.

Algorithm:

ENQUEUE(item)

1. If (queue is full)

Print “overflow”

2. if (First node insertion)

Front++

3. rear++

Queue[rear]=value

DEQUEUE()

1. If (queue is empty)

Print “underflow”

2. if(front=rear)

Front=-1 and rear=-1

3. t = queue[front]

4. front++

5. Return t

ISEMPTY()

1. If(front = -1)then

return 1

2. return 0

ISFULL()

1. If(rear = max)then

return 1

2. return 0

#include<stdio.h>

#define size 5

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

void enqueue();

void dqqueue();

void display();

int main(){

int ch;

do{

printf("enter choice: \n 1:enqueue\n 2:dqqueue\n 3:display\n 4:exit\n");

scanf("%d",&ch);

switch(ch){

case 1:

enqueue();

break;

case 2:

dqqueue();

break;

case 3:

display();

break;

case 4:

break;

default:

printf("enter valid choice\n");

}

}while(ch!=4);

return 0;

}

void enqueue(){

int x;

if(rear==size-1){

printf("queue is full\n");

}

else{

rear++;

printf("enter the value\n");

scanf("%d",&x);

queue[rear]=x;

}

}

void dqqueue(){

if(front==rear){

printf("queue is empty\n");

front=-1;

rear=-1;

}

else{

front++;

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

}

}

void display(){

if(front==rear){

printf("queue is empty\n");

}

else{

int i;

for(i=front+1; i<=rear; i++){

printf("the queue is %d\n",queue[i]);

}

}

}

OUTPUT:

Please choose the options you want to perform on queue:

1.Enqueue

2.Dequeue

3.Display

4.Exit

Conclusion :-

In conclusion, utilizing arrays to implement a Queue offers efficient FIFO data management. This approach capitalizes on arrays' constant-time element access.

However, the fixed array size can lead to potential memory inefficiencies and overflow concerns.

Careful consideration of the application's requirements is essential to determine if an array-based Queue aligns with the data dynamics.