Name :- Wahid Shaikh

Roll no :- 38 / Div 3

Date of Submission :-

Date of Performance :-

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

Code :

```
#include <stdio.h>
#define A 5
int front=-1;
int rear= -1;
int queue[A];
void enqueue(int x)
{
	if(rear == A - 1)
	{
		printf("Queue is overflow\n");
	}
	else if(front == -1 && rear == -1)
	{
		front = 0;
		rear = 0;
		queue[rear] = x;
	}
	else
	{
		rear++;
		queue[rear] = x;
	}
}
void dequeue()
```

```
{
        if(front== -1 && rear == -1)
        {
                printf("Queue is underflow\n");
        }
        else
        {
                printf("The dequeued element is %d\n", queue[front]);
                front++;
        }
}
void peek()
{
                if(front== -1 && rear == -1)
        {
                printf("Queue is underflow\n");
        }
        else
        {
        printf("The peek element is %d\n", queue[front]);
   }
}
void display()
        {
```

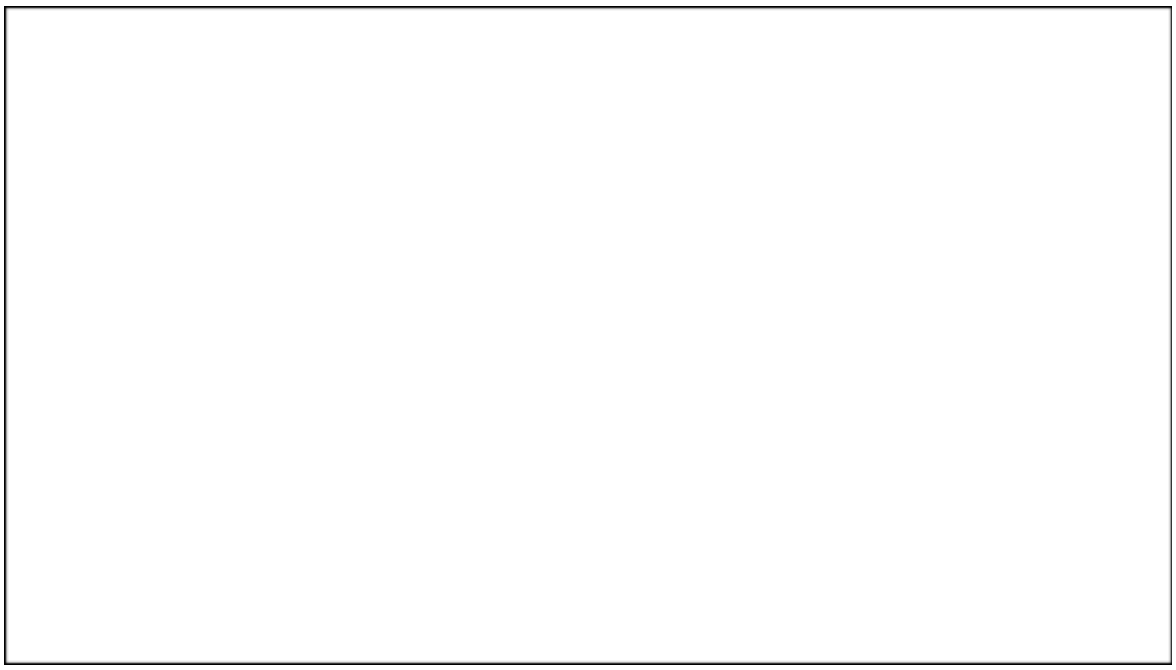
```
            int i;
            if(front==-1 && rear == -1)
            {
                  printf("Queue is underflow\n");

            }
            else
            {
            for(i= front; i< rear + 1; i++)
            {
                  printf("%d | ", queue[i]);

            }
      }
      }

void main()
{
      int i;
      for(i = 1; i>0; i++)
      {
      int choice;
      printf("\nPlease choose the options you want to perform on queue:\n1.Enqueue \n2.Dequeue \n3.Peek \n4.Display \n5.Exit\n");
```

```
scanf("%d", &choice);
if(choice == 1)
{
        int a;
        printf("Enter the element you want to enqueue:");
        scanf("%d", &a);
        enqueue(a);
}
else if(choice == 2)
{
        dequeue();
}
else if(choice == 3)
{
        peek();
}
else if(choice == 4)
{
        display();
}
else if(choice==5)
{
  break;
}
```

}

}

Output:-



```
Please choose the options you want to perform on queue:
1.Enqueue
2.Dequeue
3.Peek
4.Display
5.Exit
2
The dequeued element is 2

Please choose the options you want to perform on queue:
1.Enqueue
2.Dequeue
3.Peek
4.Display
5.Exit
3
The peek element is 0

Please choose the options you want to perform on queue:
1.Enqueue
2.Dequeue
3.Peek
4.Display
5.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.