Name	Sujal Dingankar
UID no.	DSE24100720
Experiment No.	2

AIM:	Multiple Queues in a single array.
THEORY:	A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle, which means the first element added is the first one to be removed. Queues are similar to real-world lines and can be used in many computer science applications. **The different types of queues are:** 1) Linear Queues: It stores elements in sequential order using two pointers front and rear, and once the rear pointer reaches the end, the queue is termed as full. 2) Circular queue: A type of queue where the front and rear are connected to form a circle with the help of modulo(%) operator, making operations more efficient. Circular queues are often used in operating systems to manage memory. 3) Priority Queue: It is a special type of queue where elements are dequeued based on priority rather than order of insertion. **The basic operations of a queue are:** • Enqueue: Adds an element to the back of the queue using the rear pointer, if the queue is not full. Increments the rear pointer before adding element. • Dequeue: Removes the element at the front of the queue. It increments the front pointer, then removes the item. • Peek: Gets the element at the front of the queue without removing it. • isEmpty: If front > rear pointer, then the queue is empty and it returns true. • isFull: It determines whether the queue is full or not. **Some common applications of Queue data structure are :** 1) Resource Allocation: Queues can be used to manage and allocate resources, such as printers or CPU processing time.

	2) Application of queues in Job Scheduling: For scheduled jobs, the application of queues in data structure is often used where the presence of each job is done as a process and is added to a queue. The first job is considered first, and the next job is processed once completed. 4) Data Buffering: Queues can be used for buffering data while it is being transferred between two systems. When data is received, it is added to the back of the queue, and when data is sent, it is removed from the front of the queue. In this experiment, we are implementing multiple queues in a single array. The concept can be utilized in a variety of applications where managing multiple streams of data or tasks efficiently within a limited memory space is necessary

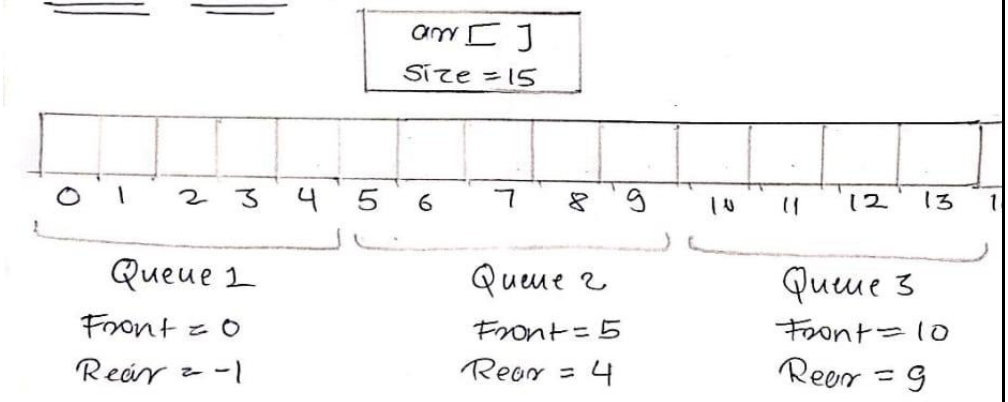
ALGORITHM:	**1. Initialize the Data:**
	• **Create an array arr[] to hold the elements of all three queues.**
	• **Create front[] and rear[] arrays to keep track of the front and rear positions of each queue.**
	  ○ **front[0], front[1], front[2] point to the start of each queue.**
	  ○ **rear[0], rear[1], rear[2] point to the last added element in each queue.**
	2. Check if a Queue is Empty:
	• **If rear[i] < front[i] for queue i, the queue is empty.**
	3. Check if a Queue is Full:
	• **Queue 1 is full if rear[0] == 4.**
	• **Queue 2 is full if rear[1] == 9.**
	• **Queue 3 is full if rear[2] == 14.**
	4. Enqueue Operation (Insert an Element):
	• **Select the queue (1, 2, or 3).**
	• **Check if the selected queue is full using the isFull() function.**
	• **If it's not full, increase the rear[i] position by 1 and insert the new element at rear[i] in the arr[].**
	5. Dequeue Operation (Remove an Element):
	• **Select the queue (1, 2, or 3).**
	• **Check if the selected queue is empty using the isEmpty() function.**
	• **If it's not empty, remove the element from the front[i] position in the arr[] and increase front[i] by 1.**
	6. Display Operation:
	• **Print all the elements from the arr[] to show the contents of all three queues.**
	7. Repeat the Menu:
	• **Continue showing options for Enqueue, Dequeue, and Display until the user chooses to exit.**

PROBLEM SOLVING:

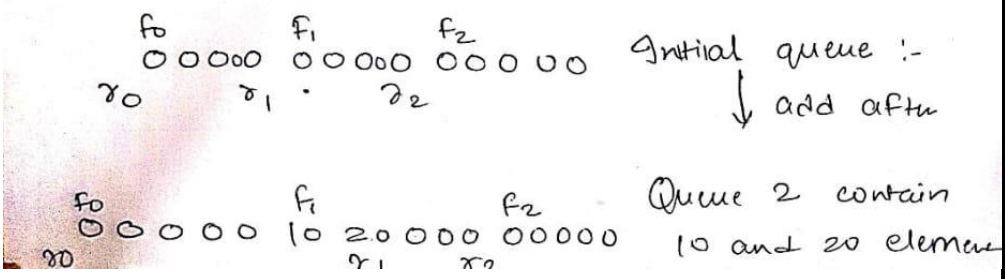
Name - Sujal Sandeep Dingankar
Topic - Multiple Queue in single array
Date → 29-09-2024.
ID - DSE24100720

* Consider an array of size 15 which we are dividing into 3 queues of size 5 each.
We are using front & rear arrays to store front & rear pointer values.

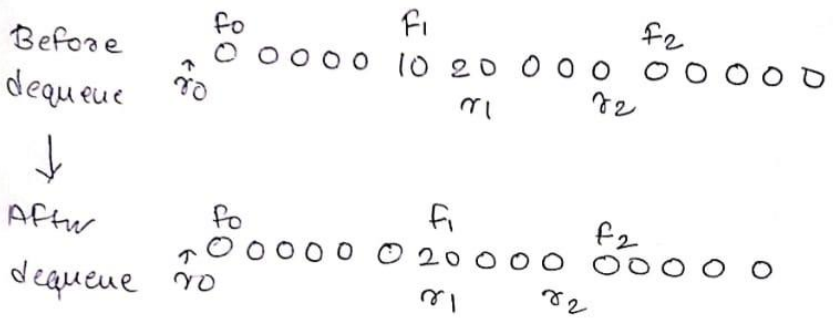
Initial condition →



• If we enqueue value 10 & 20 in second queue rear pointer increment in second queue by 1. and element are added.



* When we dequeue from Scond queue, the front pointer is used, it removes the element which it is pointing and gets incremented by 1.



Now both Front & rear pointer are pointing to the same element.

1> isEmpty() => if front > rear pointer, then queue is empty.

2> isFull() => if rear pointer exceed the fixed size allocated to Queue, then queue is full.

PROGRAM :-

```
#include <stdio.h>
#include <stdlib.h>

int size = 15;
int arr[15];
int rear[] = {-1,4,9};
int front[] = {0,5,10};

int isEmpty(int i){
   if(rear[i] < front[i]){
      return 1;
   }else{
      return 0;
   }
}

int isFull(int i){
   if(i == 0){
      return rear[i] == 4;
   }else if(i == 1){
      return rear[i] == 9;
   }else if(i == 2){
      return rear[i] == 14;
   }else{
      return 0;
   }
}

void enqueue(int data,int i){
   if(isFull(i)){
      printf("Queue is full..!!\n");
   }else{
      arr[++rear[i]] = data;
   }
}

void dequeue(int i){
```

```
    if(isEmpty(i)){
        printf("Queue is empty..\n");
    }else{
        int item = arr[front[i]];
        printf("Item which is dequeue from queue : %d \n",item);
        arr[front[i]++] = 0;
    }
}

int  main(){
    int ContinueOrNot;

    do{
        int option, queueNo, data;
        printf("Choose respective option :\n 1) Enqueue Operation..\n 2) Dequeue
Operation..\n 3) Display Queue Entire Element..\n");
        scanf("%d",&option);

        switch(option){
            case 1:
                printf("Choose queue to add element..\n");
                scanf("%d",&queueNo);
                printf("Enter data value :\n");
                scanf("%d",&data);
                enqueue(data,queueNo-1);
                break;
            case 2:
                printf("Choose queue to delete element..\n");
                scanf("%d",&queueNo);
                dequeue(queueNo - 1);
                break;
            case 3:
                printf("Display entire array of 3 queue..\n");
                for(int i=0;i<15;i++){
                    printf("%d ",arr[i]);
                }
                printf("\n");
```

```
            break;
        }

        printf("Do you want to continue operation..\n Yes then select 1 \n No then
select 0 \n");
        scanf("%d",&ContinueOrNot);
    }while(ContinueOrNot == 1);

    return 0;
}
```

RESULT :-

```
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
1
Choose queue to add element..
1
Enter data value :
55
Do you want to continue operation..
 Yes then select 1
 No then select 0
1
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
1
Choose queue to add element..
1
Enter data value :
78
Do you want to continue operation..
 Yes then select 1
 No then select 0
1
```

```
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
1
Choose queue to add element..
2
Enter data value :
56
Do you want to continue operation..
 Yes then select 1
 No then select 0
1
```

```
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
1
Choose queue to add element..
2
Enter data value :
45
Do you want to continue operation..
 Yes then select 1
 No then select 0
1
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
3
Display entire array of 3 queue..
55 78 0 0 0 56 45 0 0 0 0 0 0 0 0
Do you want to continue operation..
 Yes then select 1
 No then select 0
1
```

```
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
2
Choose queue to delete element..
2
Item which is dequeue from queue : 56
Do you want to continue operation..
 Yes then select 1
 No then select 0
1
Choose respective option :
 1) Enqueue Operation..
 2) Dequeue Operation..
 3) Display Queue Entire Element..
3
Display entire array of 3 queue..
55 78 0 0 0 0 45 0 0 0 0 0 0 0 0
```

Conclusion :-

In this experiment, I implemented a queue data structure using arrays, focusing on key operations such as enqueue, dequeue, and checking for full and empty conditions. I created three distinct queues within a single array, ensuring proper management of each queue's state. This hands-on experience enhanced my understanding of queue functionality and its practical applications in managing data efficiently.