Data Structures and Algorithms

MSc. KhangVQH

Faculty Of Information Technology

Fall - 2022

QUEUE

Terminology : Front, Rear

Operations: Insertion(),Deletion(),Display()

After Inserting five elements...

Content

- What is Queue?
- Example of Queue.
- Queue working principle
- Queue Operations
- Representation of Queue
- Coding
- output

What is a Queue?

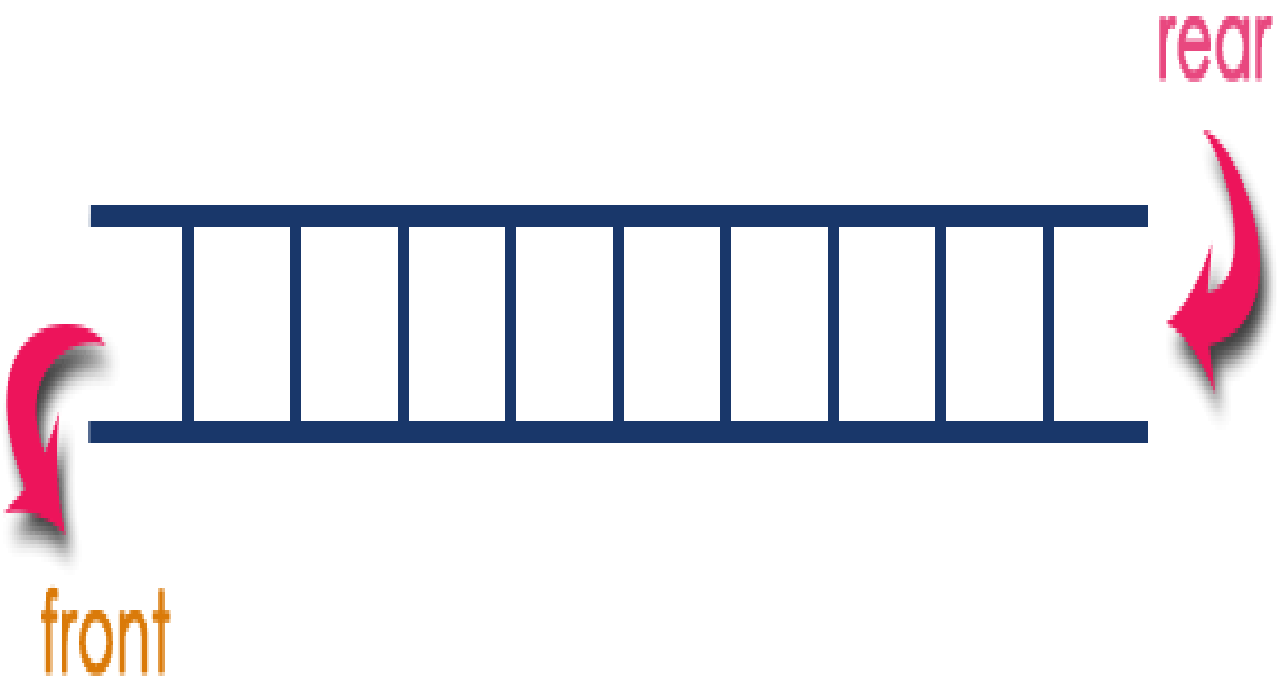
- Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends.
- In a queue data structure, adding and removing elements are performed at two different positions.
- The insertion is performed at one end and deletion is performed at another end.

- In a queue data structure, the **insertion** operation is performed at a position which is known as **'rear'**

and

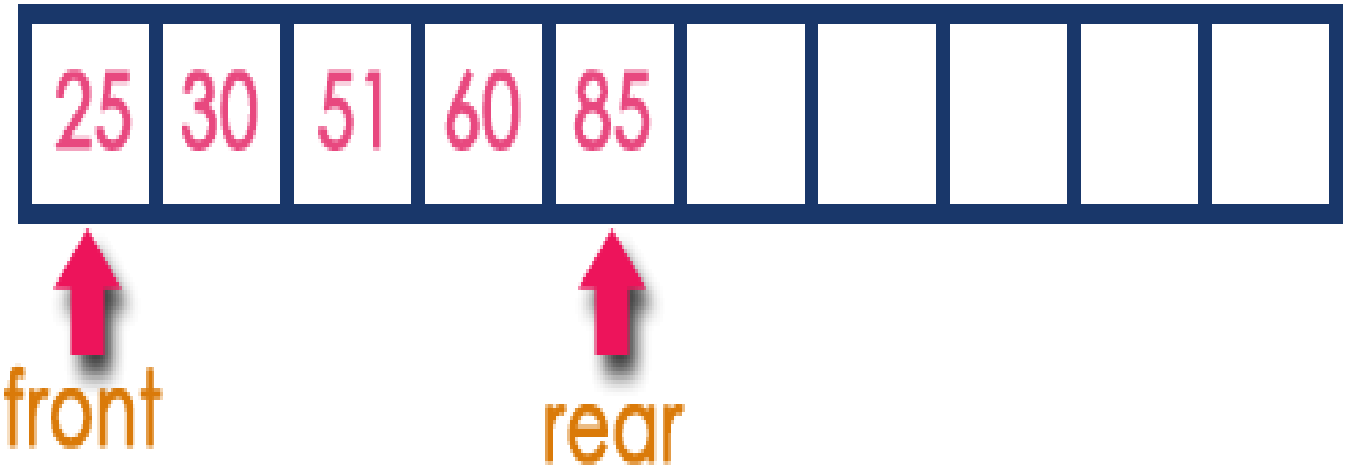
- the **deletion** operation is performed at a position which is known as **'front'.**

QUEUE



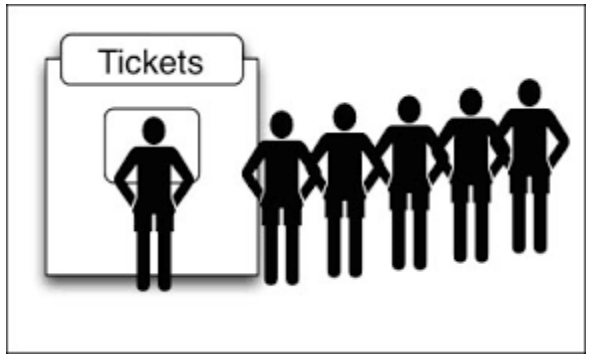
QUEUE

After Inserting five elements...



Principle

- In queue data structure, the insertion and deletion operations are performed based on **FIFO (First In First Out)** principle.



Operations on a Queue

- **enQueue(value) - (To insert an element into the queue)**
- **deQueue() - (To delete an element from the queue)**
- **display() - (To display the elements of the queue)**

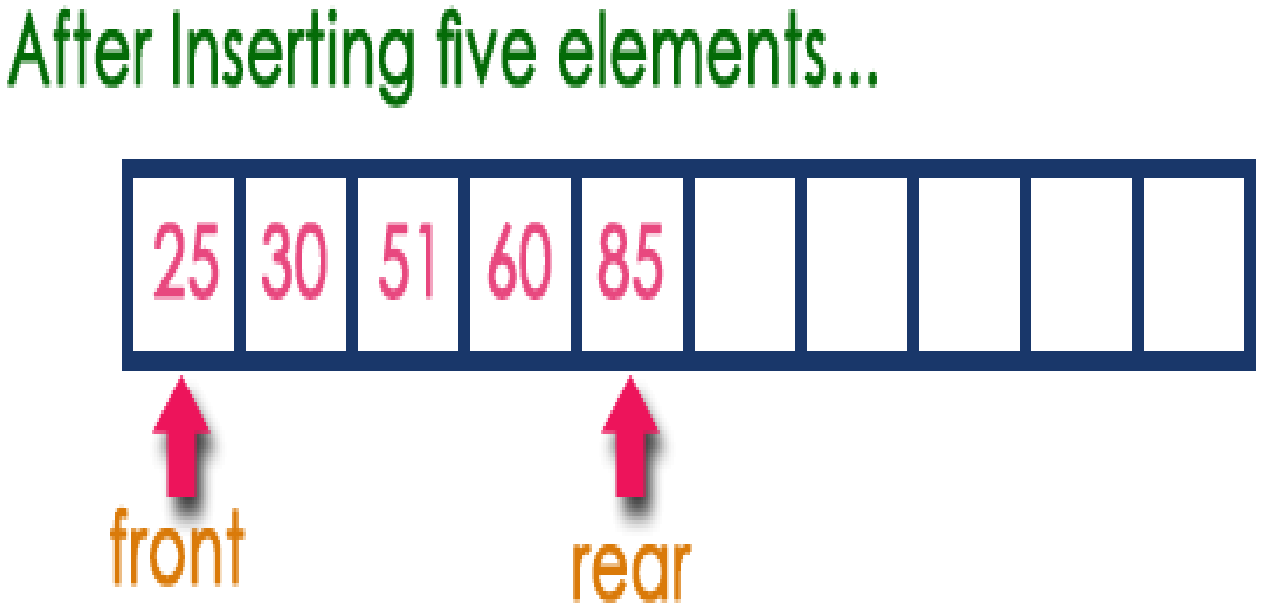
Queue data structure can be implemented in two ways.

1. **Using Array**

2. **Using Linked List**

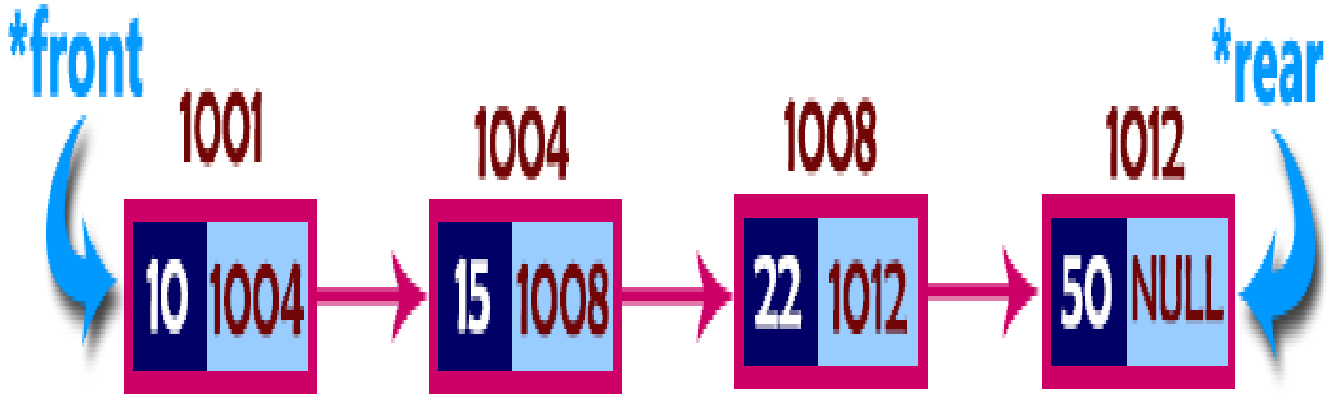
Queue data structure can be implemented in two ways.

1. **Using Array**



Queue data structure can be implemented in two ways.

2.Using Linked List



Queue Datastructure Using Array

- A queue data structure can be implemented using **one dimensional array**.

- The queue implemented using array stores only **fixed number of data values**.

Queue Datastructure Using Array

- 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.

enQueue Operation

enQueue(value) - Inserting value into the queue

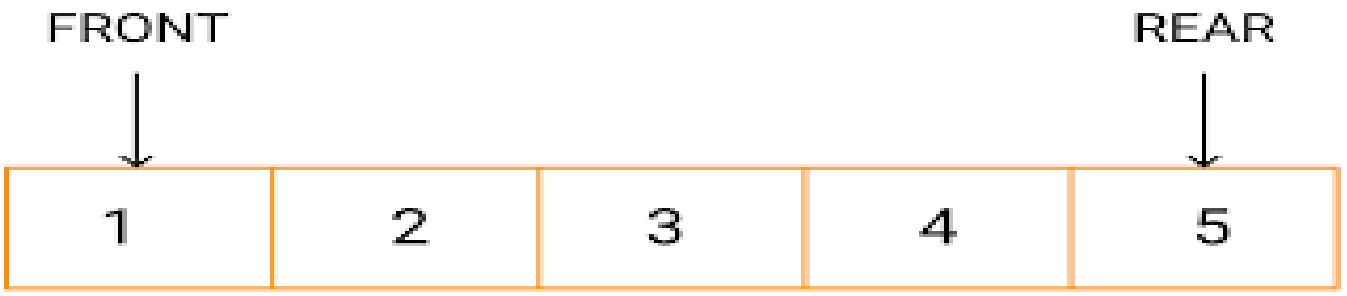
- **enQueue()** is a function used to insert a new element into the queue.
- In a queue, the **new element** is always inserted at **rear** position.
- The **enQueue()** function takes **one integer value as a parameter** and inserts that value into the queue.

enQueue(value) - Inserting value into the queue

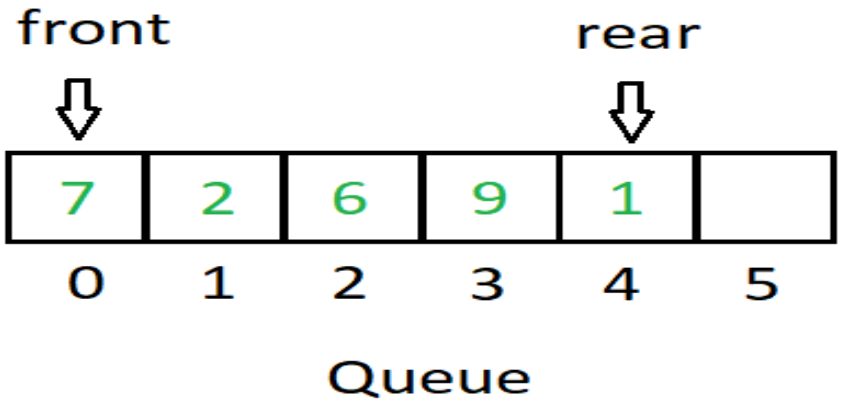
Step 1 - Check whether **queue** is **FULL**.

(rear == SIZE-1)

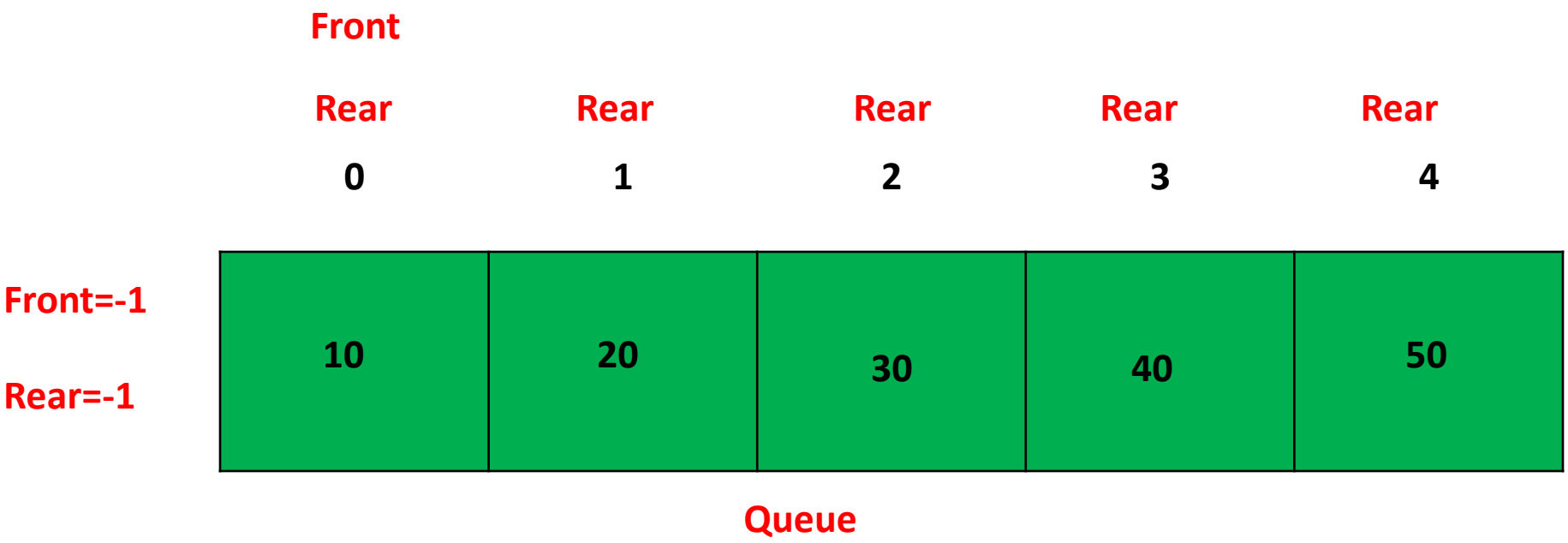
Step 2 - If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.



- **Step 3 -** If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear] = value.**



enQueue(value) - Inserting value into the queue



EnQueue(10) EnQueue(20) EnQueue(30) EnQueue(40) EnQueue(50)

DeQueue Operation

deQueue() - (To delete an element 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.

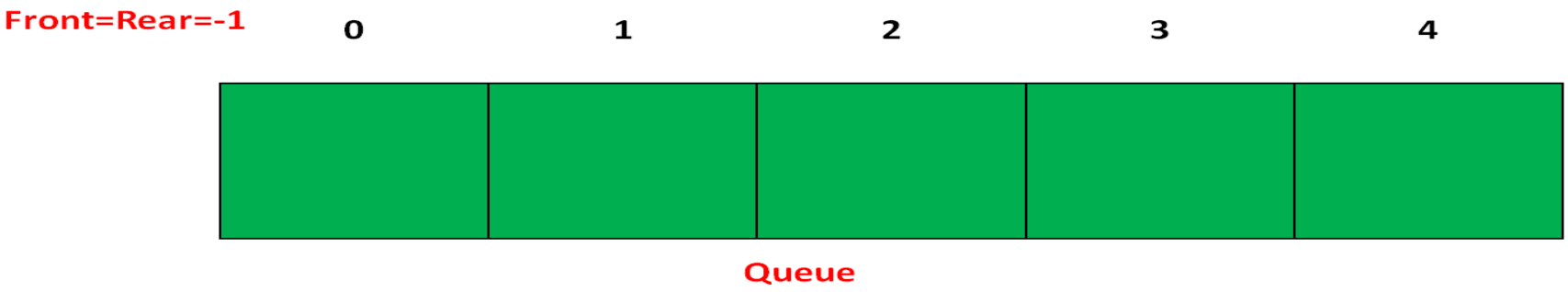
deQueue() - (To delete an element from the queue)

Step 1 - Check whether **queue** is **EMPTY**.

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

Step 2 - If it is **EMPTY**, then display

"Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function.

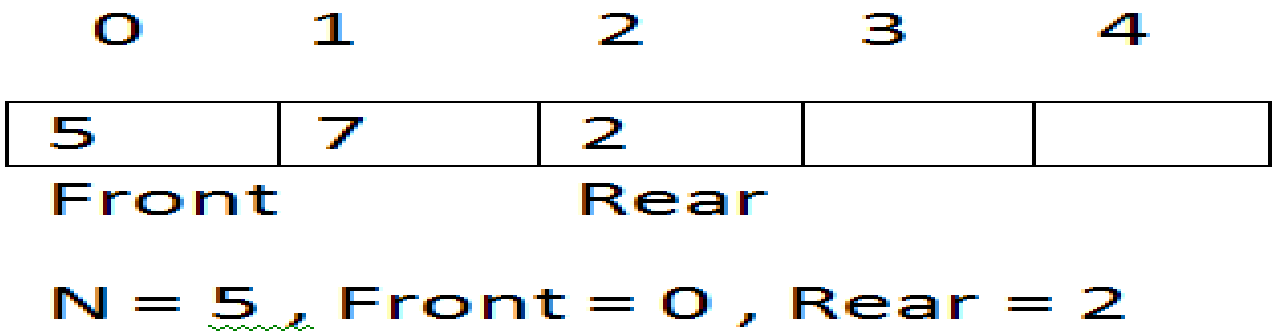


Queue is Empty, deletion not possible.

Step 3 - If it is **NOT EMPTY**,

- 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).**
- Then increment the **front** value by one

(**front ++**).



deQueue() - (To delete an element from the queue)

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

Front=Rear=-1

Front	Front	Front	Front	Front Rear
0	1	2	3	4
10	20	30	40	50

Queue

Queue is Empty, deletion not possible.

deQueue(10) deQueue(20) deQueue(30) deQueue(40) deQueue(50)

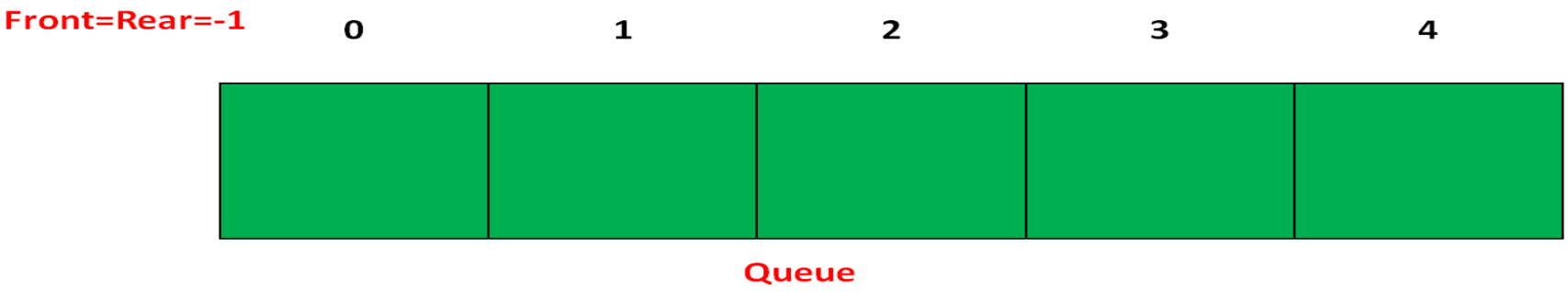
Display Operation

display() - (To display the elements of the queue)

Step 1 - Check whether **queue** is **EMPTY**.

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

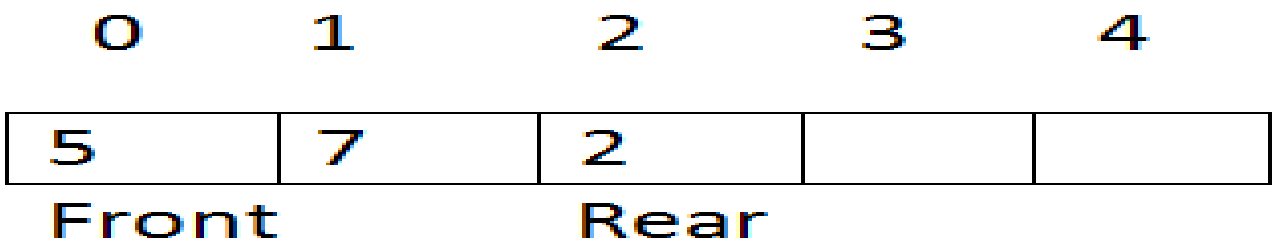
Step 2 - If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.



display() - (To display the elements of the queue)

Step 3 - If it is **NOT EMPTY**, then define an integer variable **'i'** and set **'i = front'.**

Step 4 - Display **'queue[i]'** value and increment **'i'** value by one **(i++).** Repeat the same until **'i'** value reaches to **rear (i <= rear**)



N = 5 , Front = 0 , Rear = 2

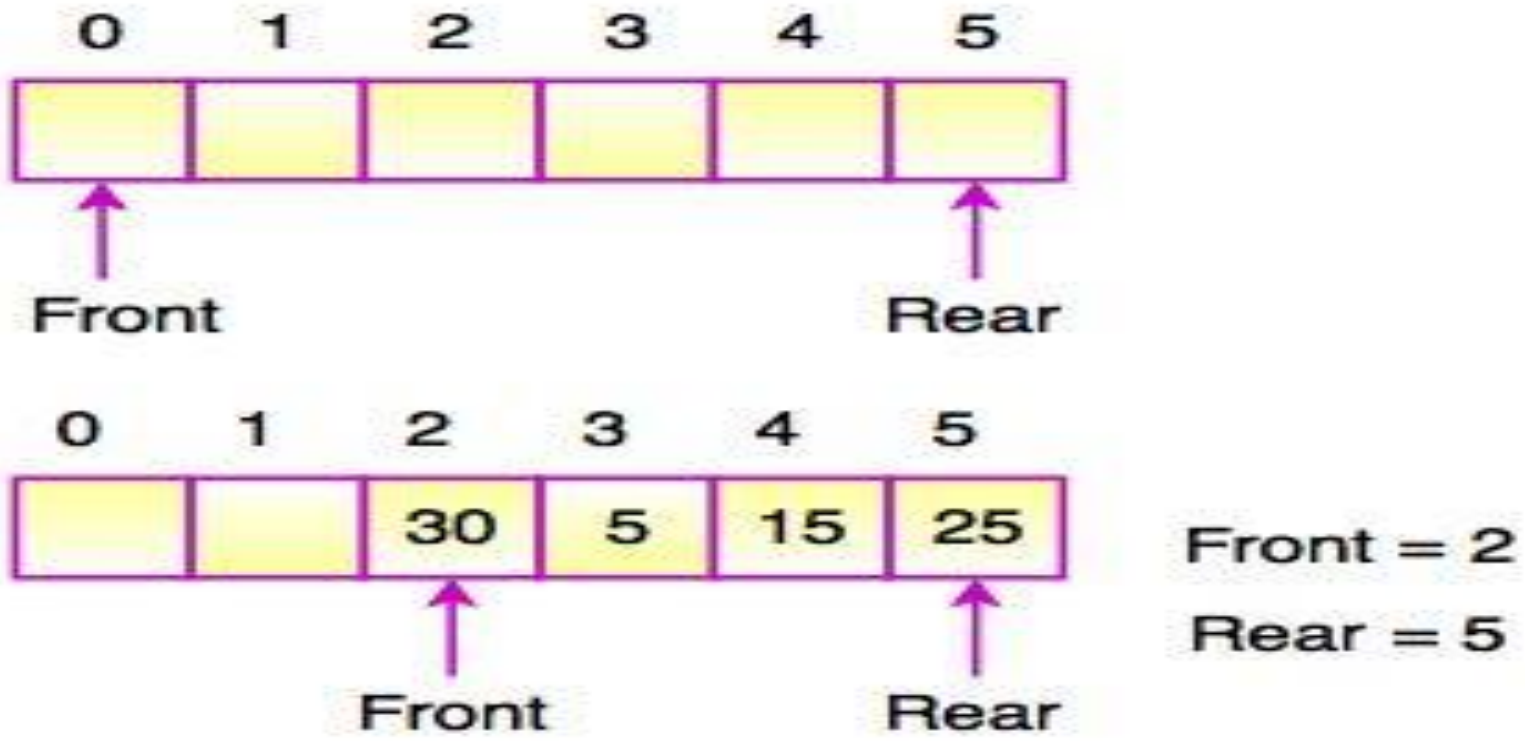


Fig. Implementation of Queue using Array

Coding for Queue

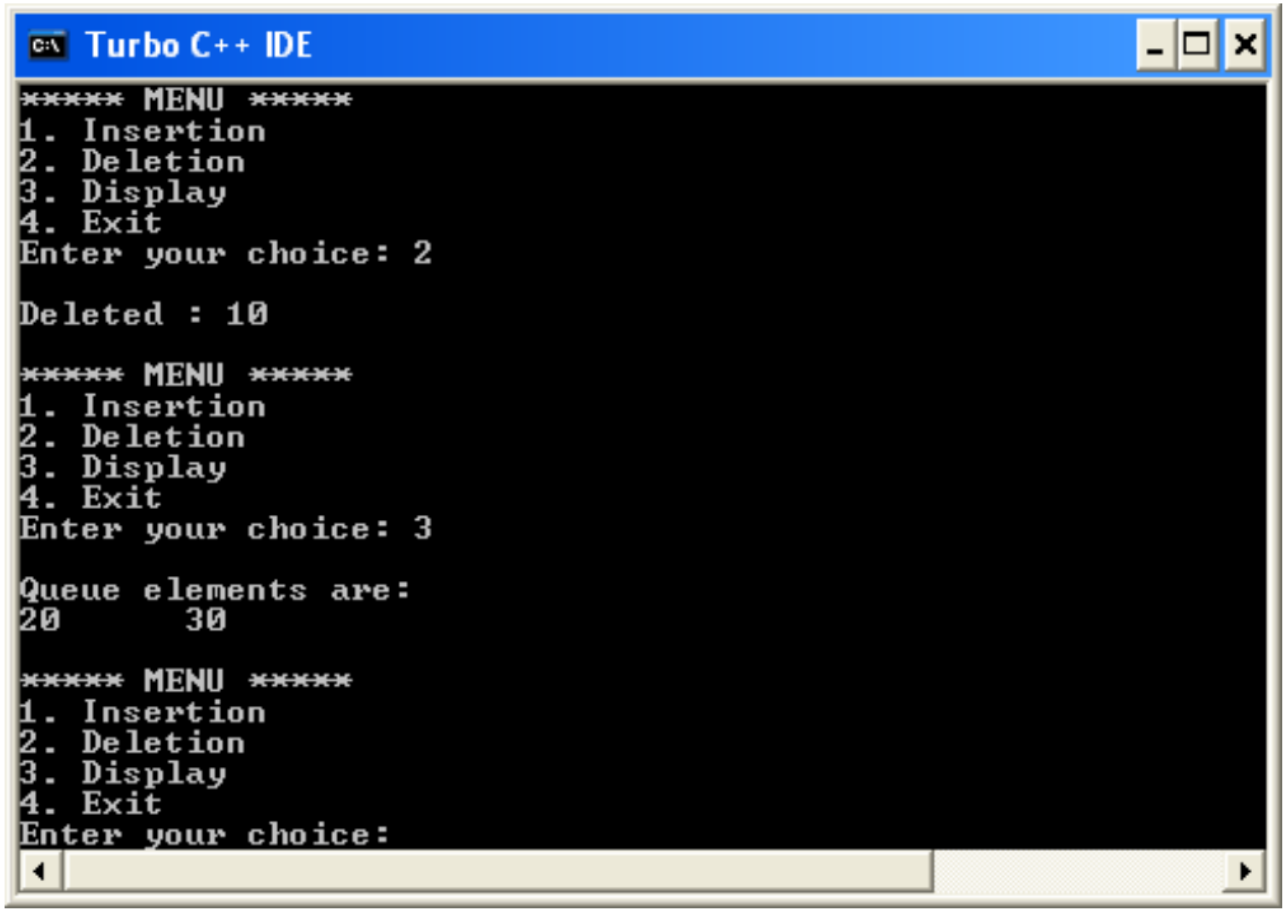
Implementation of Queue Datastructure using Array - C Programming

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10

void enQueue(int);
void deQueue();
void display();

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

void main()
{
   int value, choice;
   clrscr();
   while(1){
      printf("\n\n***** MENU *****\n");
      printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
      printf("\nEnter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("Enter the value to be insert: ");
                 scanf("%d",&value);
                 enQueue(value);
                 break;
         case 2: deQueue();
                 break;
         case 3: display();
                 break;
         case 4: exit(0);
         default: printf("\nWrong selection!!! Try again!!!");
      }
   }
}
```

```
}
void enQueue(int value){
   if(rear == SIZE-1)
      printf("\nQueue is Full!!! Insertion is not possible!!!");
   else{
      if(front == -1)
         front = 0;
      rear++;
      queue[rear] = value;
      printf("\nInsertion success!!!");
   }
}
void deQueue(){
   if(front == rear)
      printf("\nQueue is Empty!!! Deletion is not possible!!!");
   else{
      printf("\nDeleted : %d", queue[front]);
      front++;
      if(front == rear)
         front = rear = -1;
   }
}
void display(){
   if(rear == -1)
      printf("\nQueue is Empty!!!");
   else{
      int i;
      printf("\nQueue elements are:\n");
      for(i=front; i<=rear; i++)
          printf("%d\t",queue[i]);
   }
}
```

OUTPUT

Output



Thank You