4 The Queues

A queue is logically a *first in first out (FIFO or first come first serve)* linear data structure. The concept of queue can be understood by our real life problems. For example a customer come and join in a queue to take the train ticket at the end (rear) and the ticket is issued from the front end of queue. That is, the customer who arrived first will receive the ticket first. It means the customers are serviced in the order in which they arrive at the service centre.

It is a homogeneous collection of elements in which new elements are added at one end called *rear*, and the existing elements are deleted from other end called *front*.

The basic operations that can be performed on queue are

1. Insert (or add) an element to the queue (push)
2. Delete (or remove) an element from a queue (pop)

Push operation will insert (or add) an element to queue, at the rear end, by incrementing the array index. Pop operation will delete (or remove) from the front end by decrementing the array index and will assign the deleted value to a variable. Total number of elements present in the queue is front-rear+1, when implemented using arrays. Following figure will illustrate the basic operations on queue.

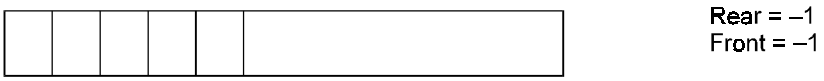

Fig. 4.1. Queue is empty.

Fig. 4.2. push(10)

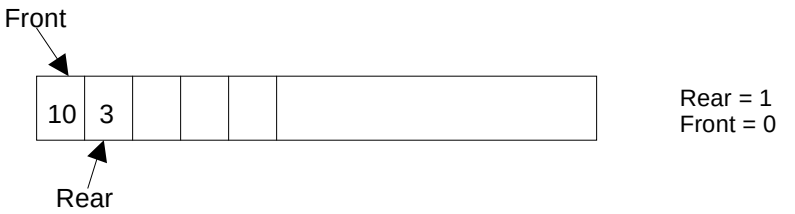

Fig. 4.3. push(3)

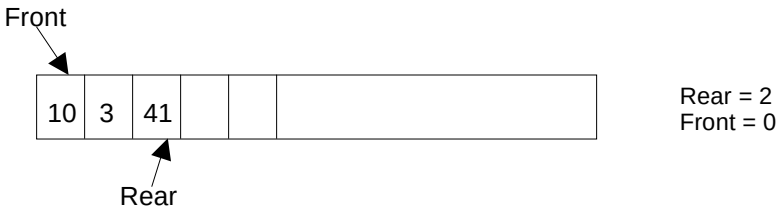

Fig. 4.4. push(41)

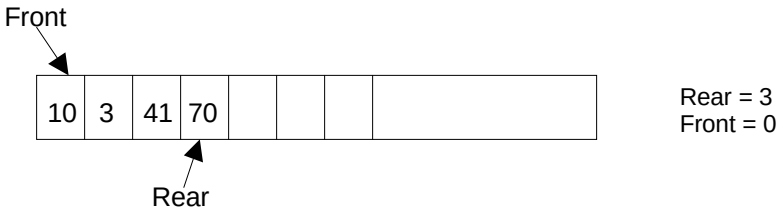

Fig. 4.5. push(70)

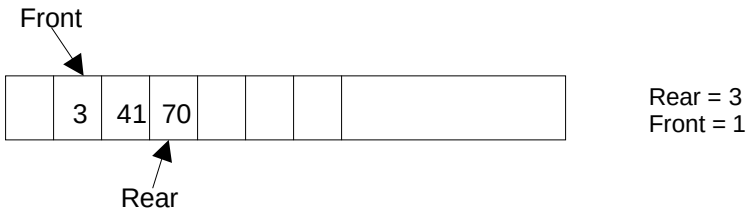

Fig. 4.6. x = pop() (*i.e.*; x = 10)

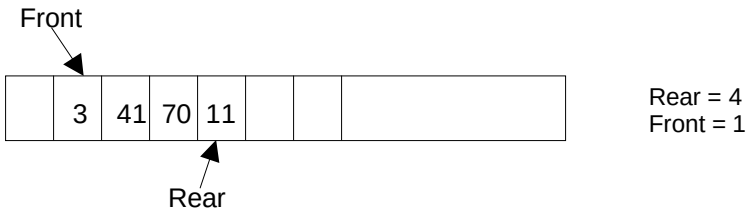

Fig. 4.7. push(11)

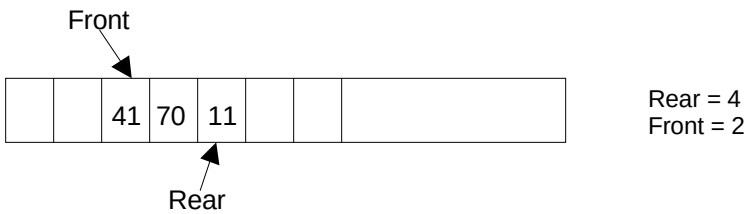

Fig. 4.8. x = pop() (*i.e.*; x = 3)

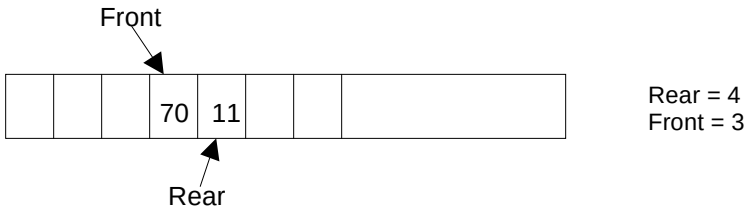

Fig. 4.9. x = pop() (*i.e.*, x = 41)

Queue can be implemented in two ways:

1. Using arrays (static)
2. Using pointers (dynamic)

Implementation of queue using pointers will be discussed in chapter 5. Let us discuss underflow and overflow conditions when a queue is implemented using arrays.

If we try to pop (or delete or remove) an element from queue when it is empty, underflow occurs. It is not possible to delete (or take out) any element when there is no element in the queue.

Suppose maximum size of the queue (when it is implemented using arrays) is 50. If we try to push (or insert or add) an element to queue, overflow occurs. When queue is full it is naturally not possible to insert any more elements

4.1. ALGORITHM FOR QUEUE OPERATIONS

Let Q be the array of some specified size say SIZE

4.1.1. INSERTING AN ELEMENT INTO THE QUEUE

1. Initialize front=0 rear = –1
2. Input the value to be inserted and assign to variable “data”
3. If (rear >= SIZE)
   (*a*) Display “Queue overflow”
   (*b*) Exit
4. Else
   (*a*) Rear = rear +1
5. Q[rear] = data
6. Exit

4.1.2. DELETING AN ELEMENT FROM QUEUE

1. If (rear< front)
   (*a*) Front = 0, rear = –1
   (*b*) Display “The queue is empty”
   (*c*) Exit

2. Else
   (*a*) Data = Q[front]
3. Front = front +1
4. Exit

PROGRAM 4.1

```
//PROGRAM TO IMPLEMENT QUEUE USING ARRAYS
//CODED AND COMPILED USING TURBO C

#include<conio.h>
#include<stdio.h>
#include<process.h>

#define MAX 50

int queue_arr[MAX];
int rear = –1;
int front = –1;

//This function will insert an element to the queue
void insert ()
{
        int added_item;
        if (rear==MAX-1)
        {
                printf("\nQueue Overflow\n");
                getch();
                return;
        }
        else
        {
                if (front==–1)  /*If queue is initially empty */
                        front=0;
                printf(“\nInput the element for adding in queue: ”);
                scanf(“%d”, &added_item);
                rear=rear+1;
                //Inserting the element
                queue_arr[rear] = added_item ;
    }
}/*End of insert()*/
```

```
//This function will delete (or pop) an element from the queue
void del()
{
        if (front == –1 || front > rear)
        {
                printf ("\nQueue Underflow\n");
                return;
        }
        else
        {
                //deleteing the element
    printf ("\nElement deleted from queue is : %d\n",
    queue_arr[front]);
                front=front+1;
        }
}/*End of del()*/

//Displaying all the elements of the queue
void display()
{
        int i;
        //Checking whether the queue is empty or not
        if (front == –1 || front > rear)
        {
                printf (“\nQueue is empty\n”);
                return;
        }
        else
        {
                printf(“\nQueue is :\n”);
                for(i=front;i<= rear;i++)
                        printf(“%d  ”,queue_arr[i]);
                printf(“\n”);
        }
}/*End of display() */

void main()
{
        int choice;
        while (1)
        {
                clrscr();
```

```
            //Menu options
            printf("\n1.Insert\n");
            printf("2.Delete\n");
            printf("3.Display\n");
            printf("4.Quit\n");
            printf("\nEnter your choice:");
            scanf("%d", & choice);
            switch(choice)
            {
            case 1 :
                  insert();
                  break;
            case 2:
                  del();
                  getch();
                  break;
            case 3:
                  display();
                  getch();
                  break;
            case 4:
                  exit(1);
            default:
                  printf ("\n Wrong choice\n");
                  getch();
            }/*End of switch*/
      }/*End of while*/
}/*End of main()*/
```

Suppose a queue Q has maximum size 5, say 5 elements pushed and 2 elements popped.

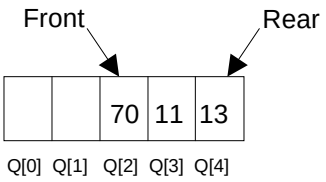

Fig. 4.10

Now if we attempt to add more elements, even though 2 queue cells are free, the elements cannot be pushed. Because in a queue, elements are always inserted at the *rear* end and hence *rear* points to last location of the queue array Q[4]. That is queue is full (overflow condition) though it is empty. This limitation can be overcome if we use circular queue.

4.2. OTHER QUEUES

There are three major variations in a simple queue. They are

1. Circular queue
2. Double ended queue (de-queue)
3. Priority queue

Priority queue is generally implemented using linked list, which is discussed in the section 5.13. The other two queue variations are discussed in the following sections.

4.3. CIRCULAR QUEUE

In circular queues the elements Q[0],Q[1],Q[2] .... Q[n – 1] is represented in a circular fashion with Q[1] following Q[n]. A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location at the queue is full.

Suppose Q is a queue array of 6 elements. Push and pop operation can be performed on circular. The following figures will illustrate the same.

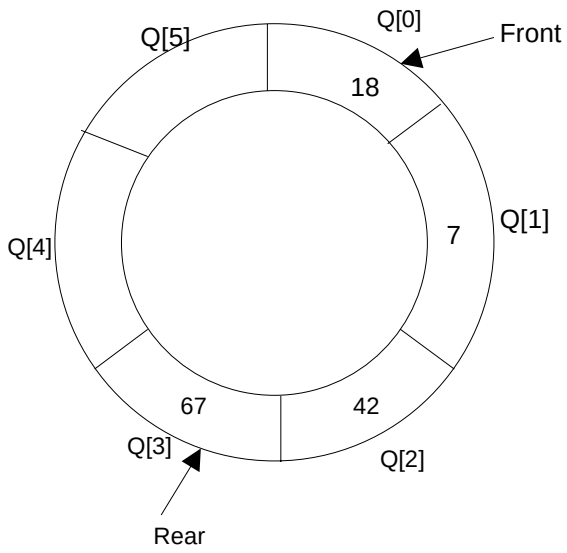

Fig. 4.11. A circular queue after inserting 18, 7, 42, 67.

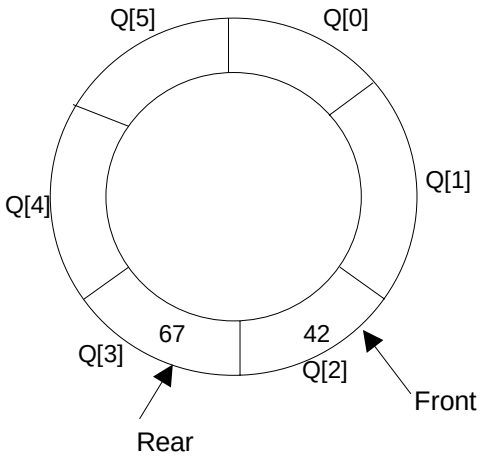

Fig. 4.12. A circular queue after popping 18, 7

After inserting an element at last location Q[5], the next element will be inserted at the very first location (*i.e.*, Q[0]) that is circular queue is one in which the first element comes just after the last element.

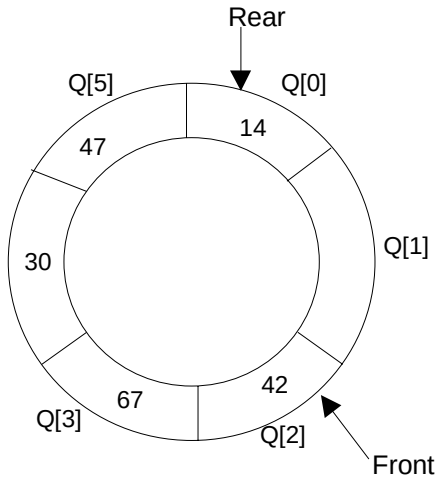

Fig. 4.13. A circular queue after pushing 30, 47, 14

At any time the position of the element to be inserted will be calculated by the relation Rear = (Rear + 1) % SIZE

After deleting an element from circular queue the position of the front end is calculated by the relation Front= (Front + 1) % SIZE

After locating the position of the new element to be inserted, *rear*, compare it with *front*. If (rear = front), the queue is full and cannot be inserted anymore.

4.3.1. ALGORITHMS

Let Q be the array of some specified size say SIZE. FRONT and REAR are two pointers where the elements are deleted and inserted at two ends of the circular queue. DATA is the element to be inserted.

Inserting an element to circular Queue

1. Initialize FRONT = – 1; REAR = 1
2. REAR = (REAR + 1) % SIZE
3. If (FRONT is equal to REAR)
   (*a*) Display "Queue is full"
   (*b*) Exit
4. Else
   (*a*) Input the value to be inserted and assign to variable "DATA"
5. If (FRONT is equal to – 1)
   (*a*) FRONT = 0
   (*b*) REAR = 0
6. Q[REAR] = DATA
7. Repeat steps 2 to 5 if we want to insert more elements
8. Exit

Deleting an element from a circular queue

1. If (FRONT is equal to – 1)
   (*a*) Display "Queue is empty"
   (*b*) Exit
2. Else
   (*a*) DATA = Q[FRONT]
3. If (REAR is equal to FRONT)
   (*a*) FRONT = –1
   (*b*) REAR = –1
4. Else
   (*a*) FRONT = (FRONT +1) % SIZE
5. Repeat the steps 1, 2 and 3 if we want to delete more elements
6. Exit

PROGRAM 4.2

```
/// PROGRAM TO IMPLEMENT CIRCULAR QUEUE USING ARRAY
//CODED AND COMPILED USING TURBO C++
```

```
#include<conio.h>
#include<process.h>
#include<iostream.h>

#define MAX 50

//A class is created for the circular queue
class circular_queue
{
        int cqueue_arr[MAX];
        int front,rear;

        public:
                //a constructor is created to initialize the variables
                circular_queue()
                {
                        front = –1;
                        rear = –1;
                }
                //public function declarations
                void insert();
                void del();
                void display();
};

//Function to insert an element to the circular queue
void circular_queue::insert()
{
        int added_item;
        //Checking for overflow condition
        if ((front == 0 && rear == MAX-1) || (front == rear +1))
        {
                cout<<"\nQueue Overflow \n";
                getch();
                return;
        }
        if (front == –1)  /*If queue is empty */
        {
                front = 0;
                rear = 0;
        }
```

```
            else
                  if (rear == MAX-1)/*rear is at last position of queue */
                        rear = 0;
                  else
                        rear = rear + 1;
            cout<<"\nInput the element for insertion in queue:";
            cin>>added_item;
            cqueue_arr[rear] = added_item;
}/*End of insert()*/

//This function will delete an element from the queue
void circular_queue::del()
{
            //Checking for queue underflow
            if (front == –1)
            {
                  cout<<"\nQueue Underflow\n";
                  return;
            }
            cout<<"\nElement deleted from queue is:"<<cqueue_arr[front]<<"\n";
            if (front == rear) /* queue has only one element */
            {
                  front = –1;
                  rear = –1;
            }
            else
                  if(front == MAX-1)
                        front = 0;
                  else
                        front = front + 1;
}/*End of del()*/

//Function to display the elements in the queue
void circular_queue::display()
{
            int front_pos = front,rear_pos = rear;
            //Checking whether the circular queue is empty or not
            if (front == –1)
            {
                  cout<<"\nQueue is empty\n";
                  return;
            }
```

```
		//Displaying the queue elements
		cout<<"\nQueue elements:\n";
		if(front_pos <= rear_pos )
			while(front_pos <= rear_pos)
			{
				cout<<cqueue_arr[front_pos]<<", ";
				front_pos++;
			}
		else
		{
			while(front_pos <= MAX-1)
			{
				cout<<cqueue_arr[front_pos]<<", ";
				front_pos++;
			}
			front_pos = 0;
			while(front_pos <= rear_pos)
			{
				cout<<cqueue_arr[front_pos]<<", ";
				front_pos++;
			}
		}/*End of else*/
		cout<<"\n";
}/*End of display() */

void main()
{
		int choice;
		//Creating the objects for the class
		circular_queue co;
		while(1)
		{
			clrscr();
			//Menu options
			cout <<"\n1.Insert\n";
			cout <<"2.Delete\n";
			cout <<"3.Display\n";
			cout <<"4.Quit\n";
			cout <<"\nEnter your choice: ";
			cin>>choice;

			switch(choice)
```

```
                {
                case 1:
                        co.insert();
                        break;
                case 2 :
                        co.del();
                        getch();
                        break;
                case 3:
                        co.display();
                        getch();
                        break;
                case 4:
                        exit(1);
                default:
                        cout<<"\nWrong choice\n";
                        getch();
                }/*End of switch*/
        }/*End of while*/
}/*End of main()*/
```

4.4. DEQUES

A deque is a homogeneous list in which elements can be added or inserted (called push operation) and deleted or removed from both the ends (which is called pop operation). ie; we can add a new element at the rear or front end and also we can remove an element from both front and rear end. Hence it is called Double Ended Queue.

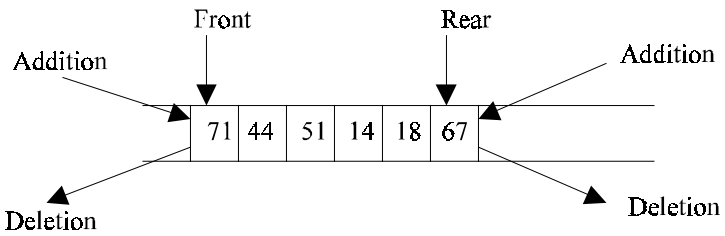

Fig. 4.14. A deque

There are two types of deque depending upon the restriction to perform insertion or deletion operations at the two ends. They are

1. Input restricted deque
2. Output restricted deque

An input restricted deque is a deque, which allows insertion at only 1 end, rear end, but allows deletion at both ends, rear and front end of the lists.

An output-restricted deque is a deque, which allows deletion at only one end, front end, but allows insertion at both ends, rear and front ends, of the lists.

The possible operation performed on deque is

1. Add an element at the rear end
2. Add an element at the front end
3. Delete an element from the front end
4. Delete an element from the rear end

Only 1st, 3rd and 4th operations are performed by input-restricted deque and 1st, 2nd and 3rd operations are performed by output-restricted deque.

4.4.1. ALGORITHMS FOR INSERTING AN ELEMENT

Let Q be the array of MAX elements. *front* (or left) and *rear* (or *right*) are two array index (pointers), where the addition and deletion of elements occurred. Let DATA be the element to be inserted. Before inserting any element to the queue *left* and *right* pointer will point to the – 1.

INSERT AN ELEMENT AT THE RIGHT SIDE OF THE DE-QUEUE

1. Input the DATA to be inserted
2. If ((left == 0 && right == MAX–1) || (left == right + 1))
   (*a*) Display "Queue Overflow"
   (*b*) Exit
3. If (left == –1)
   (*a*) left = 0
   (*b*) right = 0
4. Else
   (*a*) if (right == MAX –1)
      (*i*) left = 0
   (*b*) else
      (*i*) right = right+1
5. Q[right] = DATA
6. Exit

INSERT AN ELEMENT AT THE LEFT SIDE OF THE DE-QUEUE

1. Input the DATA to be inserted
2. If ((left == 0 && right == MAX–1) || (left == right+1))
   (*a*) Display "Queue Overflow"
   (*b*) Exit

3. If (left == – 1)
   (*a*) Left = 0
   (*b*) Right = 0
4. Else
   (*a*) if (left == 0)
      (*i*) left = MAX – 1
   (*b*) else
      (*i*) left = left – 1
5. Q[left] = DATA
6. Exit

4.4.2. ALGORITHMS FOR DELETING AN ELEMENT

Let Q be the array of MAX elements. *front* (or *left*) and *rear* (or *right*) are two array index (pointers), where the addition and deletion of elements occurred. DATA will contain the element just deleted.

DELETE AN ELEMENT FROM THE RIGHT SIDE OF THE DE-QUEUE

1. If (left == – 1)
   (*a*) Display "Queue Underflow"
   (*b*) Exit
2. DATA = Q [right]
3. If (left == right)
   (*a*) left = – 1
   (*b*) right = – 1
4. Else
   (*a*) if(right == 0)
      (*i*) right = MAX-1
   (*b*) else
      (*i*) right = right-1
5. Exit

DELETE AN ELEMENT FROM THE LEFT SIDE OF THE DE-QUEUE

1. If (left == – 1)
   (*a*) Display "Queue Underflow"
   (*b*) Exit
2. DATA = Q [left]
3. If(left == right)
   (*a*) left = – 1
   (*b*) right = – 1

4. Else
   (*a*) if (left == MAX-1)
      (*i*) left = 0
   (*b*) Else
      (*i*) left = left +1
5. Exit

PROGRAM 4.3

```
//PROGRAM TO IMPLEMENT INPUT AND OUTPUT
//RESTRICTED DE-QUEUE USING ARRAYS
//CODED AND COMPILED USING TURBO C

#include<conio.h>
#include<stdio.h>
#include<process.h>

#define MAX 50

int deque_arr[MAX];
int left = –1;
int right = –1;

//This function will insert an element at the
//right side of the de-queue
void insert_right()
{
        int added_item;
        if ((left == 0 && right == MAX-1) || (left == right+1))
        {
                printf ("\nQueue Overflow\n");
                getch();
                return;
        }
        if (left == –1)  /* if queue is initially empty */
        {
                left = 0;
                right = 0;
        }
        else
```

```
            if(right == MAX-1)  /*right is at last position of queue */
                  right = 0;
            else
                  right = right+1;
            printf("\n Input the element for adding in queue: ");
            scanf ("%d", &added_item);
            //Inputting the element at the right
            deque_arr[right] = added_item ;
}/*End of insert_right()*/

//Function to insert an element at the left position
//of the de-queue
void insert_left()
{
            int added_item;
            //Checking for queue overflow
            if ((left == 0 && right == MAX-1) || (left == right+1))
            {
                  printf ("\nQueue Overflow \n");
                  getch();
                  return;
            }
            if (left == –1)/*If queue is initially empty*/
            {
                  left = 0;
                  right = 0;
            }
            else
            if (left== 0)
                  left = MAX –1;
            else
                  left = left-1;
            printf("\nInput the element for adding in queue:");
            scanf ("%d", &added_item);
            //inputting at the left side of the queue
            deque_arr[left] = added_item ;
}/*End of insert_left()*/

//This function will delete an element from the queue
//from the left side
void delete_left()
{
```

```
        //Checking for queue underflow
        if (left == –1)
        {
                printf("\nQueue Underflow\n");
                return;
        }
        //deleting the element from the left side
        printf ("\nElement deleted from queue is: %d\n",deque_arr[left]);
        if(left == right) /*Queue has only one element */
        {
                left = –1;
                right=–1;
        }
        else
                if (left == MAX-1)
                        left = 0;
                else
                        left = left+1;
}/*End of delete_left()*/

//Function to delete an element from the right hand
//side of the de-queue
void delete_right()
{
        //Checking for underflow conditions
        if (left == –1)
        {
                printf("\nQueue Underflow\n");
                return;
        }
        printf("\nElement deleted from queue is : %d\n",deque_arr[right]);
        if(left == right) /*queue has only one element*/
        {
                left = –1;
                right=–1;
        }
        else
                if (right == 0)
                        right=MAX-1;
                else
                        right=right-1;
}/*End of delete_right() */
```

```
//Displaying all the contents of the queue
void display_queue()
{
        int front_pos = left, rear_pos = right;
        //Checking whether the queue is empty or not
        if (left == –1)
        {
                printf (“\nQueue is empty\n”);
                return;
        }
        //displaying the queue elements
        printf ("\nQueue elements :\n");
        if ( front_pos <= rear_pos )
        {
                while(front_pos <= rear_pos)
                {
                        printf (“%d ”,deque_arr[front_pos]);
                        front_pos++;
                }
        }
        else
        {
                while(front_pos <= MAX-1)
                {
                        printf(“%d ”,deque_arr[front_pos]);
                        front_pos++;
                }
                front_pos = 0;
                while(front_pos <= rear_pos)
                {
                        printf (“%d ”,deque_arr[front_pos]);
                        front_pos++;
                }
        }/*End of else */
        printf (“\n”);
}/*End of display_queue() */

//Function to implement all the operation of the
//input restricted queue
void input_que()
{
        int choice;
        while(1)
```

```
        {
                clrscr();
                //menu options to input restricted queue
                printf ("\n1.Insert at right\n");
                printf ("2.Delete from left\n");
                printf ("3.Delete from right\n");
                printf ("4.Display\n");
                printf ("5.Quit\n");
                printf ("\nEnter your choice : ");
                scanf ("%d",&choice);

                switch(choice)
                {
                 case 1:
                        insert_right();
                        break;
                 case 2:
                        delete_left();
                        getch();
                        break;
                 case 3:
                        delete_right();
                        getch();
                        break;
                 case 4:
                        display_queue();
                        getch();
                        break;
                 case 5:
                        exit(0);
                 default:
                        printf("\nWrong choice\n");
                        getch();
                }/*End of switch*/
        }/*End of while*/
}/*End of input_que() */

//This function will implement all the operation of the
//output restricted queue
void output_que()
{
        int choice;
```

```
        while(1)
        {
                clrscr();
                //menu options for output restricted queue
                printf ("\n1.Insert at right\n");
                printf ("2.Insert at left\n");
                printf ("3.Delete from left\n");
                printf ("4.Display\n");
                printf ("5.Quit\n");
                printf ("\nEnter your choice:");
                scanf ("%d",&choice);

                switch(choice)
                {
                 case 1:
                        insert_right();
                        break;
                 case 2:
                        insert_left();
                        break;
                 case 3:
                        delete_left();
                        getch();
                        break;
                 case 4:
                        display_queue();
                        getch();
                        break;
                 case 5:
                        exit(0);
                 default:
                        printf("\nWrong choice\n");
                        getch();
                }/*End of switch*/
        }/*End of while*/
}/*End of output_que() */

void main()
{
        int choice;
        clrscr();
        //Main menu options
        printf ("\n1.Input restricted dequeue\n");
```

```
            printf ("2.Output restricted dequeue\n");
            printf ("Enter your choice:");
            scanf ("%d",&choice);

            switch(choice)
            {
             case 1:
                    input_que();
                    break;
             case 2:
                    output_que();
                    break;
             default:
                    printf("\nWrong choice\n");
            }/*End of switch*/
}/*End of main()*/
```

If we analyze the algorithms in this chapter the time needed to add or delete a data is constant, *i.e.* time complexity is of order O(1).

4.5. APPLICATIONS OF QUEUE

1. Round robin techniques for processor scheduling is implemented using queue.
2. Printer server routines (in drivers) are designed using queues.
3. All types of customer service software (like Railway/Air ticket reservation) are designed using queue to give proper service to the customers.