CHAPTER 2
DATA STRUCTURES: STACK & QUEUE

2.9 Queue

- **Def:**
  - "A data structure, in which elements can be **added** at **one** end and **removed** from the **other** end."
- **Example:**
  - Persons standing in a "Line" at ticket-window.
- **Characteristic:**
  - **FIFO: F**irst **I**n **F**irst **O**ut.
  - First inserted element (item) comes out First. (Explain this with an example.)
- **Representation:**
  - **Vertical:** (Provide figure as discussed in class.)
  - **Horizontal:** (Provide figure as discussed in class.)
- **Operations:**
  - **Insert:** Adds / Inserts an element in a queue.
  - **Delete:** Removes / Deletes an element from a queue.
- **Implementation:**
  - By using **Array** (Static Memory Allocation)
  - By using **Linked List** / **Pointer** (Dynamic Memory Allocation)

[Note: Use "DECROI" to remember above points.]

2.10 Queue Operations: Insert

- **Def:**
  - "Process of inserting an element into queue."
- **Explanation:**
  - (Provide explanation as discussed in class.)
- **Queue Overflow:**
  - "Situation, arising during **Insert** operation, when Queue is **Full**."

- **Function:**

```
void    qinsert ( int    x )
{
        // Check for an Overflow…
    if ( rear == MAX-1 )
    {
        printf ( " Queue Overflow…\n ");
        exit ( 0 );
    }

        // Update rear pointer…
    rear ++ ;

        // Store an element at rear end of queue…
    queue [ rear ] = x;

        // Update front pointer…
    if ( front == -1 )
        front = 0 ;
}
```

- **Algorithm:**
  - **QINSERT (X)**
    - [Inserts given element 'X' in a queue.]
  - **Variables:**

    i) QUEUE: Array having MAX elements.

    ii) MAX: No. of maximum elements in a queue.

    iii) FRONT: Pointer to track front end of a queue.

    iv) REAR: Pointer to track rear end of a queue.

    v) X: Element to be inserted in a queue
  - **Steps:**
    - **Step-1:** [Check for queue Overflow.]

```
IF      (REAR = MAX-1)      THEN
        WRITE ('Queue Overflow…')
        EXIT
END IF
```

    - **Step-2:** [Update rear pointer.]

```
REAR  ←  REAR + 1
```

- **Step-3:** [Store an element at rear end of a queue.]

  QUEUE [REAR] ← X

- **Step-4:** [Update front pointer.]

  IF (FRONT = -1) THEN
  FRONT ← 0
  END IF

- **Step-5:** [Finish]

  RETURN

2.11 Queue Operations: Delete

- **Def:**
  - "Process of removing an element from a queue."
- **Explanation:**
  - (Provide explanation as discussed in class.)
- **Queue Underflow:**
  - "Situation, arising during **Delete** operation, when Queue is **Empty**."
- **Function:**

```
int     qdelete ( )
{
        int temp;
                // Check for an Underflow…
        if ( front == -1 )
        {
                printf ( " Queue Underflow…\n ");
                exit ( 0 );
        }
                // Read an element to be deleted…
        temp = queue [ front ];
                // Update front and rear pointer…
        if ( front == rear )
                front = rear = -1;
        else
                front ++;
                // Return an element to be deleted…
        return ( temp );
}
```

- **Algorithm:**
  - **QDELETE ( )**
    - [Removes / Deletes an element from queue.]
  - **Variables:**

    i) QUEUE: Array having MAX elements.

    ii) MAX: No. of maximum elements in a queue.

    iii) FRONT: Pointer to track front end of a queue.

    iv) REAR: Pointer to track rear end of a queue.

    v) TEMP: Stores an element to be deleted.
  - **Steps:**
    - **Step-1:** [Check for queue Underflow.]

```
IF      ( FRONT = -1 )  THEN
        WRITE ('Queue Underflow...')
        EXIT
END IF
```

    - **Step-2:** [Read an element to be deleted.]

```
TEMP  ←  QUEUE [ FRONT ]
```

    - **Step-3:** [Update front and rear pointer.]

```
IF ( FRONT = REAR )  THEN
        FRONT  ←  -1
        REAR  ←  -1
ELSE
        FRONT  ←  FRONT + 1
END IF
```

    - **Step-4:** [Return an element to be removed.]

```
RETURN ( TEMP )
```

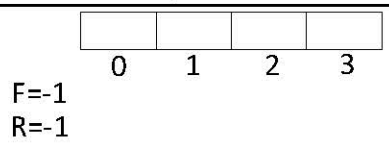
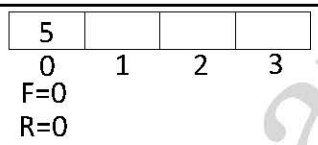
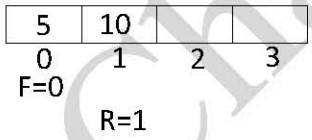
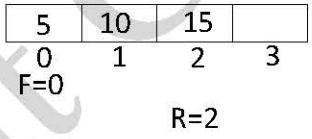
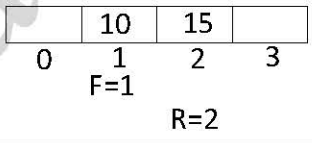
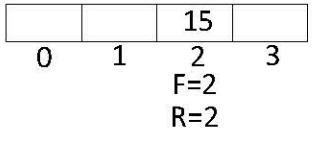
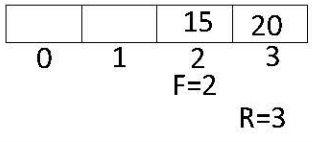
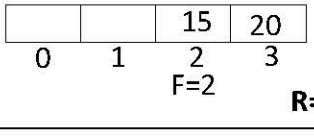
2.12 Implementation of Queue

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

#define MAX 5

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

void main()
{
        int choice, n;

        void qinsert(int);
        int qdelete();

        while(1)
        {
                printf("\n1. Insert Operation.\n");
                printf("2. Delete Operation.\n");
                printf("3. Exit.\n");

                printf("\n Enter Ur Choice : ");
                scanf("%d",&choice);

                switch(choice)
                {
                        case 1:
                                printf(" Enter element to be inserted : ");
                                scanf("%d",&n);
                                qinsert(n);
                                break;

                        case 2:
                                n = qdelete();
                                printf(" Deleted element is : %d\n",n);
                                break;

                        case 3:
                                printf("\n Program terminated successfully...");
                                exit(0);

                        default:
                                printf("\n Invalid choice...\n");
                }
        }
}
```

```
// define qinsert function...
void qinsert(int x)
{
        // check for an Overflow...
    if (rear == MAX-1)
    {
            printf("\n Queue Overflow...\n");
            exit(0);
    }

        // update rear pointer...
    rear ++ ;
        // store an element at rear end of queue...
    queue [ rear ] = x;

        // update front pointer...
    if ( front == -1 )
        front = 0 ;
}

// define qdelete function...
int qdelete()
{
    int temp;

        // check for an Underflow...
    if ( front == -1 )
    {
        printf("\n Queue Underflow...\n");
        exit(0);
    }

        // read an element to be deleted...
    temp = queue [ front ];

        // update front and rear pointer...
    if ( front == rear )
        front = rear = -1;
    else
        front ++;

        // return an element to be deleted...
    return ( temp );
}
```

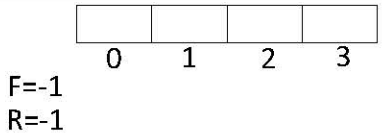
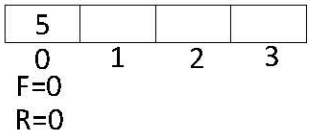
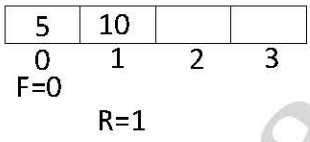
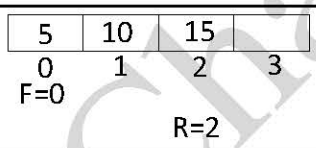
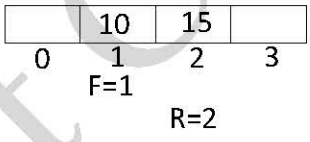
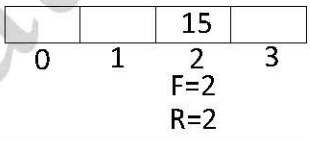
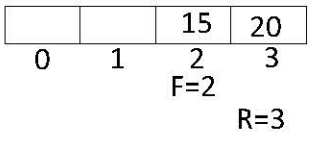
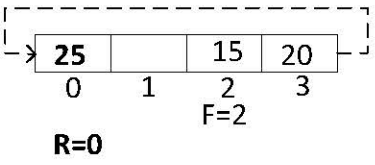
2.13 Circular Queue

- **Def:**
  - "A queue, in which elements are added and removed in a circular fashion / manner, is called Circular Queue."
- **Drawback of Linear (Simple) Queue:**
  - Linear queue may get overflow even though some locations at front end are empty.
  - Example:

No.	Operation	Linear Queue Status	Result
1.	Initialization		Empty Queue.
2.	Insert **5**		**5** Inserted.
3.	Insert **10**		**10** Inserted.
4.	Insert **15**		**15** Inserted.
5.	Delete		**5** Deleted.
6.	Delete		**10** Deleted.
7.	Insert **20**		**20** Inserted.
8.	Insert **25**		**Queue Overflow.**

  - Here, two locations at front end are empty; though they can't be used to insert 25.
  - This results in **memory wastage.**

- **Advantage of Circular Queue:**
  - In circular queue, empty locations at front end can be utilized to insert new elements in circular manner.
  - Example:

No.	Operation	Circular Queue Status	Result
1.	Initialization		Empty Queue.
2.	Insert **5**		**5** Inserted.
3.	Insert **10**		**10** Inserted.
4.	Insert **15**		**15** Inserted.
5.	Delete		**5** Deleted.
6.	Delete		**10** Deleted.
7.	Insert **20**		**20** Inserted.
8.	Insert **25**		**25** Inserted.

  - Here, 25 can be inserted at the front end of a circular queue, which was not possible in linear queue.
  - This results in **efficient utilization** of memory.

- **Representation of Circular Queue:**
  - (Give circular figure of queue as discussed in class.)

2.14 **Circular Queue Operations: Insert (Not in Syllabus)

- **Function:**

```
void    cqinsert        ( int    x )
{
            // Update rear pointer…
        if ( rear == MAX-1 )
            rear = 0;
        else
            rear ++ ;

            // Check for an Overflow…
        if ( front == rear )
        {
            printf ( " Circular Queue Overflow…\n ");
            exit ( 0 );
        }

            // Store an element at rear end of queue…
        cqueue [ rear ] = x;

            // Update front pointer…
        if ( front == -1 )
            front = 0 ;
}
```

- **Algorithm:**

  ❖ **CQINSERT (X)**

  - [Inserts given element 'X' in a queue.]

  ❖ **Variables:**

  i) **CQUEUE:** Array having MAX elements.
  ii) **MAX:** No. of maximum elements in a circular queue.
  iii) **FRONT:** Pointer to track front end of a circular queue.
  iv) **REAR:** Pointer to track rear end of a circular queue.
  v) **X**: Element to be inserted in a circular queue

  ❖ **Steps:**

  - **Step-1:** [Update rear pointer.]

```
IF      (REAR = MAX-1)      THEN
        REAR ← 0
ELSE
        REAR  ←  REAR + 1
END IF
```

- **Step-2:** [Check for circular queue Overflow.]

```
IF      (FRONT == REAR)      THEN
        WRITE ( ' Circular Queue Overflow... ' )
        EXIT
END IF
```

- **Step-3:** [Store an element at rear end of a queue.]

```
REAR [ CQUEUE ]  ← X
```

- **Step-4:** [Update front pointer.]

```
IF      ( FRONT = -1 )  THEN
        FRONT  ←  0
END IF
```

- **Step-5:** [Finish]

```
RETURN
```

2.15 **Circular Queue Operations: Delete (Not in Syllabus)

- **Function:**

```
int     cqdelete ( )
{
        int temp;
                // Check for an Underflow...
        if ( front == -1 )
        {
                printf ( " Circular Queue Underflow...\n ");
                exit ( 0 );
        }
                // Read an element to be deleted...
        temp = cqueue [ front ];
                // Update front and rear pointer...
        if ( front == rear )
                front = rear = -1;
        else if ( front == MAX - 1 )
                front = 0;
        else
                front ++;
```

```
                    // Return an element to be deleted...
            return ( temp );
        }
```

- **Algorithm:**
  - ❖ **CQDELETE ( )**
    - [Removes / Deletes an element from queue.]
  - ❖ **Variables:**
    - **i) CQUEUE:** Array having MAX elements.
    - **ii) MAX:** No. of maximum elements in a circular queue.
    - **iii) FRONT:** Pointer to track front end of a circular queue.
    - **iv) REAR:** Pointer to track rear end of a circular queue.
    - **v) TEMP**: Stores an element to be deleted.
  - ❖ **Steps:**
    - **Step-1:** [Check for queue Underflow.]

```
IF      ( FRONT = -1 )  THEN
        WRITE ( ' Circular Queue Underflow... ' )
        EXIT
END IF
```

    - **Step-2:** [Read an element to be deleted.]

```
TEMP  ←  FRONT [ CQUEUE ]
```

    - **Step-3:** [Update front and rear pointer.]

```
IF ( FRONT = REAR ) THEN
        FRONT  ←  -1
        REAR  ←  -1
ELSE IF ( FRONT = MAX – 1 ) THEN
        FRONT ← 0
ELSE
        FRONT  ←  FRONT + 1
END IF
```

    - **Step-4:** [Return an element to be removed.]

```
RETURN ( TEMP )
```

● ● ●