Queues

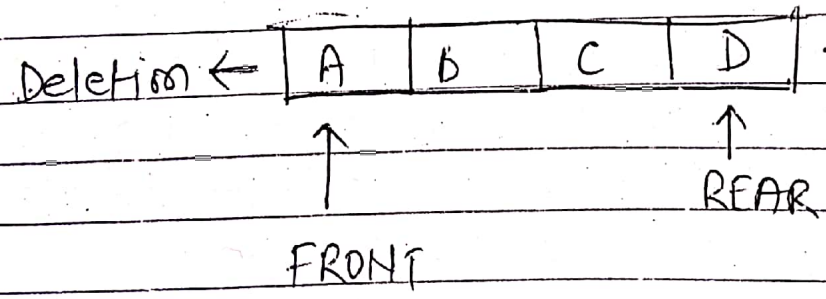
→ A queue is a linear data structure in which insertion can take place at only one end called rear and deletion can take place at the other end called front.

→ FRONT(Q) denotes the front of queue (Q)
REAR (Q) denotes the end of queue (Q)

Suppose $Q = [Q_1, Q_2, \ldots Q_n]$

$Front(Q) = Q_1$
$Rear(Q) = Q_n$

→ A queue may be represented as :-



→ The information in such a list is processed in the same order as it was received i.e. FIFO.

Application of queue

1. Real life example
   - Waiting in a line
   - waiting on hold for tec support

2. Applications related to computer science

- Task waiting for the printing
- Job scheduling (e.g. Round Robin algorithm for cpu allocation)
- cpu scheduling in operating system uses queue

The queue as ADT

(i) create(queue):

- The operation create(q) creates an empty queue with name q. Thus, Noel(create(q))=0.

(ii) Isempty(queue):

- Isempty operation determines whether a queue is empty or not.
- Is result of is of boolean type.
- Isempty(q) is true if queue is empty and false otherwise.
- Thus Isempty(create(q)) = true

(iii) Isfull(queue):

- This operation checks whether the queue is full or not.
- If it is not full, insert operation can be performed.

(iv) Insert (elemet, queue):

- Inserting an element to queue is called insert operation.

- Insert (E, Q) adds element E in a queue Q.

- Thus Rear (insert (E, Q)) = E

- Isempty (insert (E, Q)) is false

(v) Remove (queue)

- To remove the front element from a queu is called remove operation.

- remove (create(Q)) gives error condition because queue Q is empty.

Algorithm for inserting element in a queue

Steps of Algorithm

Step 1: If (Rear = n-1)
a) write "QUEUE FULL"
b) goto step3

Step 2: If front = Rear = -1 then
a) set front ← 0

b) set Rear ← 0
c) set Q[Rear] ← val
else

a) set Rear ← Rear+1
b) set Q[Rear] ← val

step3: stop

(implement of "insert" operation in Queue

```
qinsert(int q[], int val)
{
    if (rear==n-1)
    {
        printf("Queue full\n");
    }
else if (rear==-1 && front==-1)
{
    front = 0;
    rear = 0;
    q[rear]= val;
}
else
{
    rear++;
    q[rear]=val;
}
}
```

Algorithm for removal operation in QUEUE

①

step 1 Repeat stages 2 to 4 until front >=0
step 2 set item = queue[front]
step 3 if front == real
set front = -1
set Rear = -1
Else
front = front+1
Endif

step 4 print, NO deleted is, item
step 5 print, " queue is empty"

②

step 1: if (front = -1)
(a) write "QUEUE empty"
(b) goto step 3

step 2: If (front = Rear) then
(a) set val ← Q[front]
(b) set front ← -1
(c) set Rear ← -1

else
(a) set val ← Q[front]
(b) set Front ← front+1

step 3: stop

c implementation of delete operation in queue

```
void delete()
{
    if(front==-1 || front>rear)
    {
      printf("queue underflow\n");
    }

    else
    {
```

```
void delete()
{
    int item;
    if(front!=-1)
    {
        item= queue[front];
        if(front=rear)
        {
          front= -1;
          Rear = -1;
        }
        else
          front = front+1;
        printf(" No deleted is =%d ", item);
    }
    else
        printf("Queue is empty");
}
```

Array implementation of queue.

```
#include<stdio.h>
#include<conio.h>
#define SIZE 4
void qinsert(int q[], int item);
void qdelete(int q[]);
void display(int q[]);
int front, rear;
void main()
{

int q[SIZE], val, element, choice;
clrscr();
front = -1;
rear = -1;
printf("MAIN MENU\n");
printf("for Insert operation enter choice=1\n");
printf("for Delete operation enter choice=2\n");
printf("for exit enter choice=3\n");

do
{
    printf("Enter your choice:");
    scanf("%d", &choice);
    if(choice==1)
    {

    Printf("Enter the element to insert\n");
    scanf("%d", &element);
```

```
qinsert(q, element);
}
else if (choice == 2)
{
qdelete(q);
}
}while(choice==1||choice==2);
getch();
}
void qinsert (int q[], int val)
{
if(rear == SIZE-1)
{
printf("Queue is full\n");
}
else
{
if(rear == -1 && front == -1)
{
front=0;
rear=0;
q[rear]= val;
}
```

```
    else
    {
      rear++;
      q[rear] = val;
    }

    display(q);
  }
}

void qdelete (int q[])
{
  int val;
  if (rear == -1)
  {
    printf("queue is empty\n");
  }
  else if (rear == front)
  {

    val = q[front];
    front = -1;
    rear = -1;
    printf("The deleted item is = %d\n", val);

  }
  else
```

```
{
  val = q[front];
  front++;
  printf("Deleted item is = %d\n", val);
  display();
}
}

void display (int q[])
{
  int i;
  printf("Queue elements are...\n");
  printf("front --->");
  for (i=front ; i<=rear ; i++)
    printf("%d ", q[i]);
    printf("<--- Rear");
  printf("\n");
}
```

O/p: MAIN MENU
for insert operation enter choice=1
for delete operation enter choice = 2
for exit enter choice=3
Enter your choice : 1
Enter the elements to insert
11
Queue elements are...
front ---> 11 <--- Rear

Enter your choice: 1
enter the element to insert
22
queue elements are
front ---> 11 22 <--- Rear

8. Implementation of queue using array

```
#include<conio.h>
#include<stdio.h>
#define MAX 50
 void insert();
 void delete();
 void display();
 int queue_array[MAX];
 int rear = -1;
 int front = -1;
void main()
 {
    int choice;
  while(1)
  {
  printf("1. Insert element to queue\n");
  printf("2. Delete element from queue\n");
  printf("3. Display all elements of queue\n");
  printf("4. Quit\n");
  printf("Enter your choice\n");
  scanf("%d", &choice);
  switch(choice)
  {
    case 1:
```

```
        insert();
        break;

        Case 2:
        delete();
        break;

        Case 3:
        exit();
        break;

        default:
        printf("Wrong choice \n");
    } /* end of switch case */

} /* End of while */
} /* end of main()*/

void insert()
{
    int add_item;
    if (rear = MAX-1)
    printf("Queue overflow \n");
    else
    {
        if(front == -1)
        /* If queue is initially empty */
        front = 0;
        printf("Insert the element in queue : ");
        scanf("%d", &add_item);
```

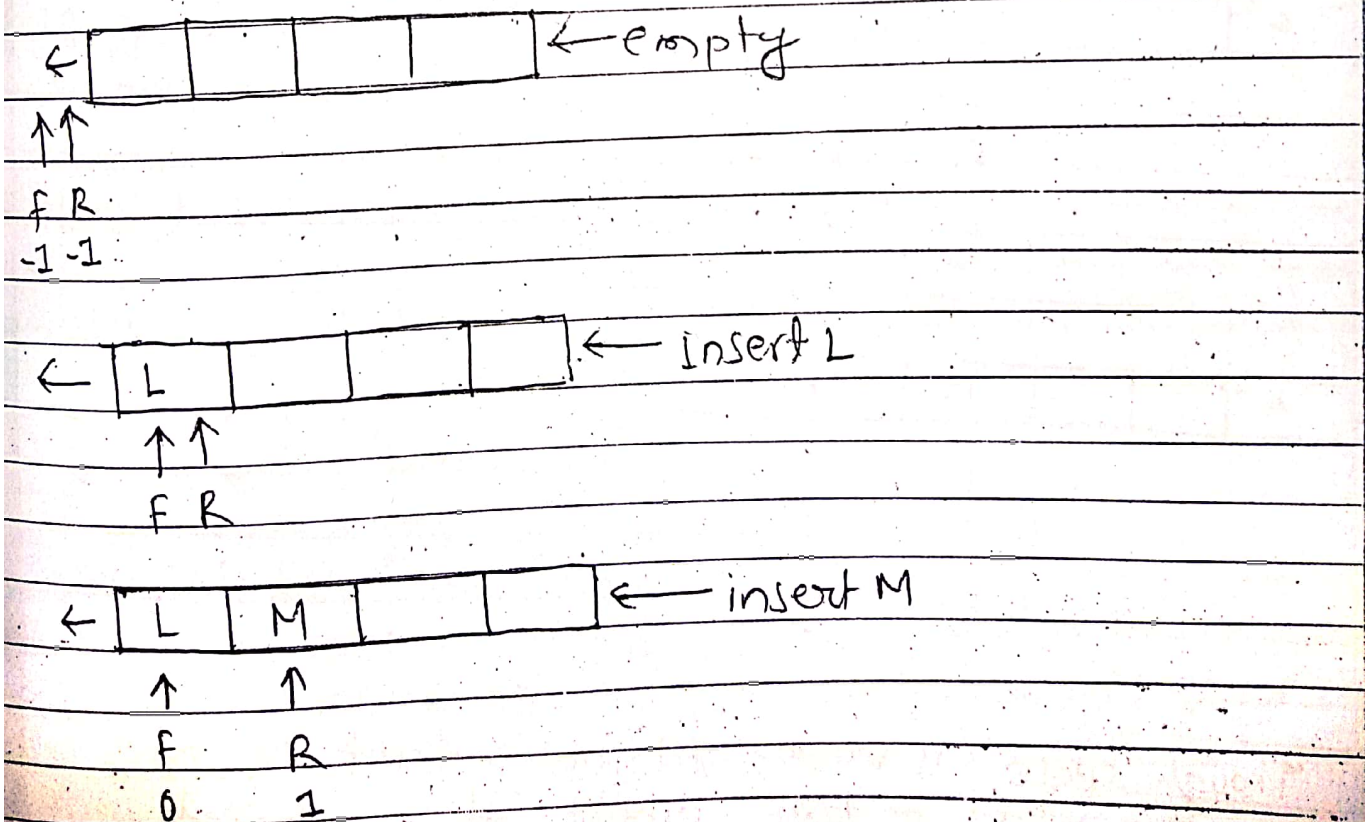
```
        rear=rear+1;
        queue_array[rear]=add_item;
    }
}/* end of insert() */

Void delete()
{
    if(front==-1 || front>rear)
    {
        printf("queue underflow\n");
    }
    else
    {
        printf("element deleted from queue is: %d\n",
                        queue_array[front]);
        front=front+1;
    }
}/* end of delete()*/

void display()
{
    int i;
    if (front==-1)
        Printf("Queue is empty\n")
    else
    {
        Printf("Queue is :\n");
```

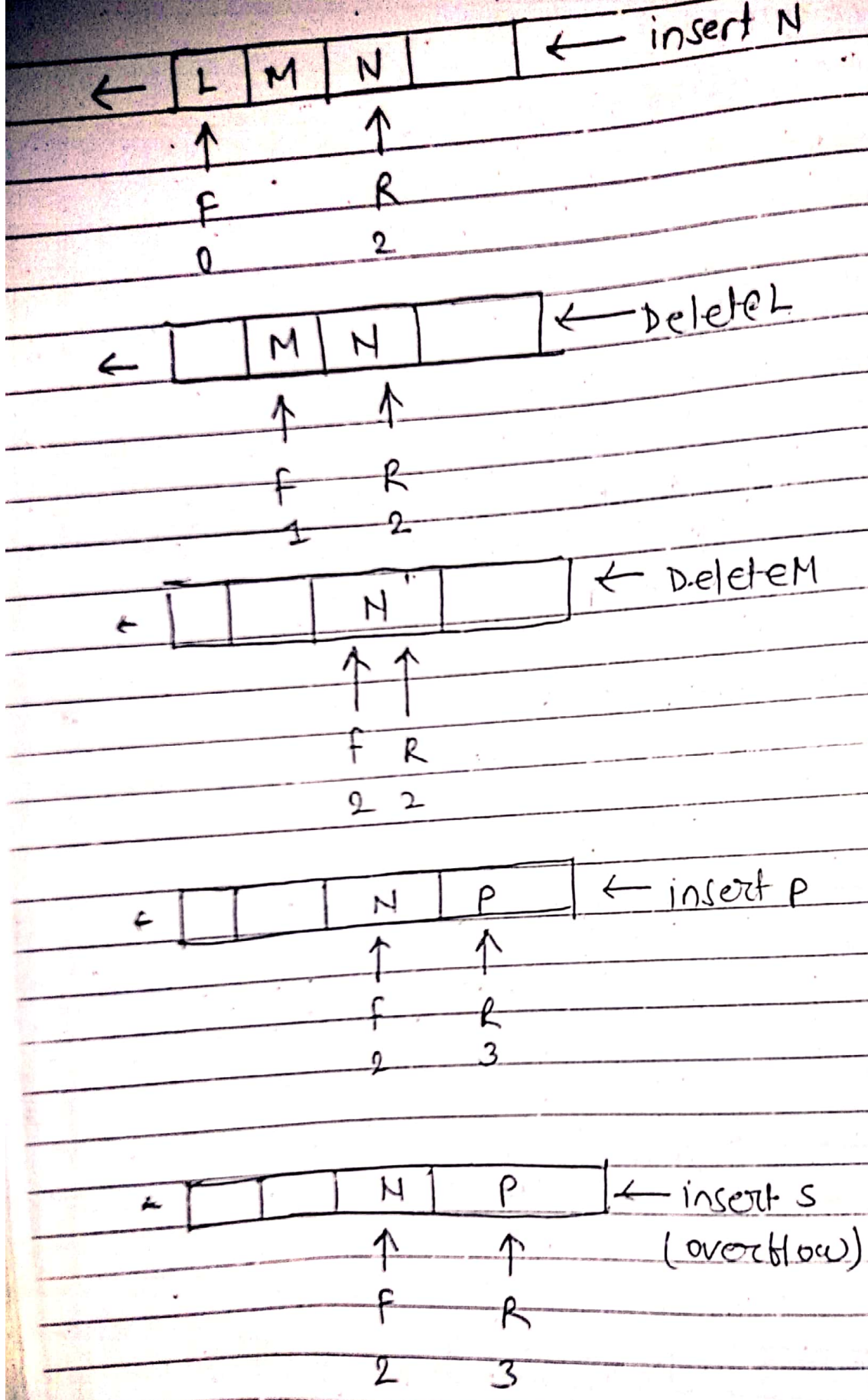
```
for (i=front ; i<=rear ; i++)
    printf ("%d", queue_array[i]);
    printf("\n");
}
}/* End of display() */
```

Disadvantage of linear queue

The only disadvantage of linear queue is: "if the last position of the queue is occupied, insertion of any more element is not possible even though some position are vacant towards the front of the queue"

- Consider an example where the size of the queue is for elements.



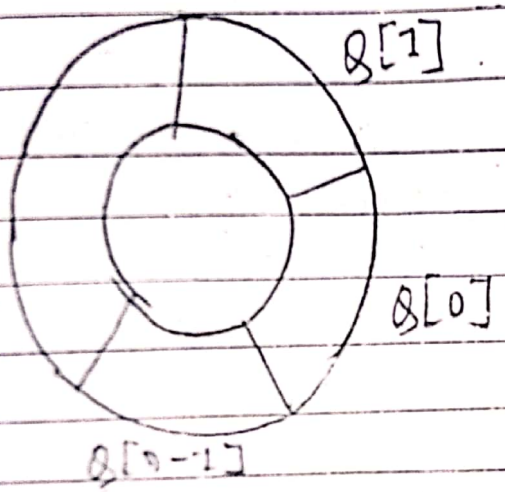


Though space is vacant yet no element can be inserted. Thus a large amount of memory

would be required to accommodate the elements in such representation.

Circular QUEUE

It is linear data structure in which front and rear more in a circular way.



- In a circular queue the fist element Q[0] is followed by Q[n-1].

steps of algorithm

step1 : If(front=0 and Rear=n-1) or (front=rear+1)
(a) write "CIRCULAR QUEUE FULL"
(b) goto step4

step2: If(fRont = Rear = -1) then
(a) set front ← 0
(b) set Rear ← 0
(c) set CQ[Rear] ← val

Step 3 : If (Rear = n-1) then
(a) set Rear ← 0
(b) set CQ[Rear] ← val
(c) else
(a) set Rear ← Rear+1
(b) Set CQ[Rear] ← val

Step 4 : stop

Algorithm to remove element from a circula queue.

steps of algorithm

step 1 : If (front = Rear = -1) then
a) write "circular QUEUE EMPTY"
b) goto step 4

step 2 : if (front = Rear) then

a) Set val ← CQ[Front]
b) Set front ← -1
c) set Rear ← -1

step 3 : If (front = n-1) then
a) setval ← CQ[front]
b) Set front ← 0
else
b) Set val ← CQ[front]
b) Set front ← front+1

Step4 : stop

Note:

The c programming language provides a Keyword called typdef, which you can use to give a type a new name. Following is an example to define a term BYTE for one-byte numbers-

typdef unsigned char BYTE

After this type definition, the identifier BYTE can be used as an abbreviation for the type unsigned char, for example.

~~Byte~~ BYTE b1, b2;

By convention, uppercase letters are used for these definition to remind the user that the type name is really a symbolic abbreviation, but you can use lower case, as follows -

typdef unsigned char byte;

You can use typdef to give a name to your user defined data types as well. for example, you can use typedef with structure to define a new datatype and then use that datatype to define structure variables directly as follows: