UNIT - IV

Introduction to Data structure

→ A data structure is a specialized format for organizing, processing, retrieving and storing data.

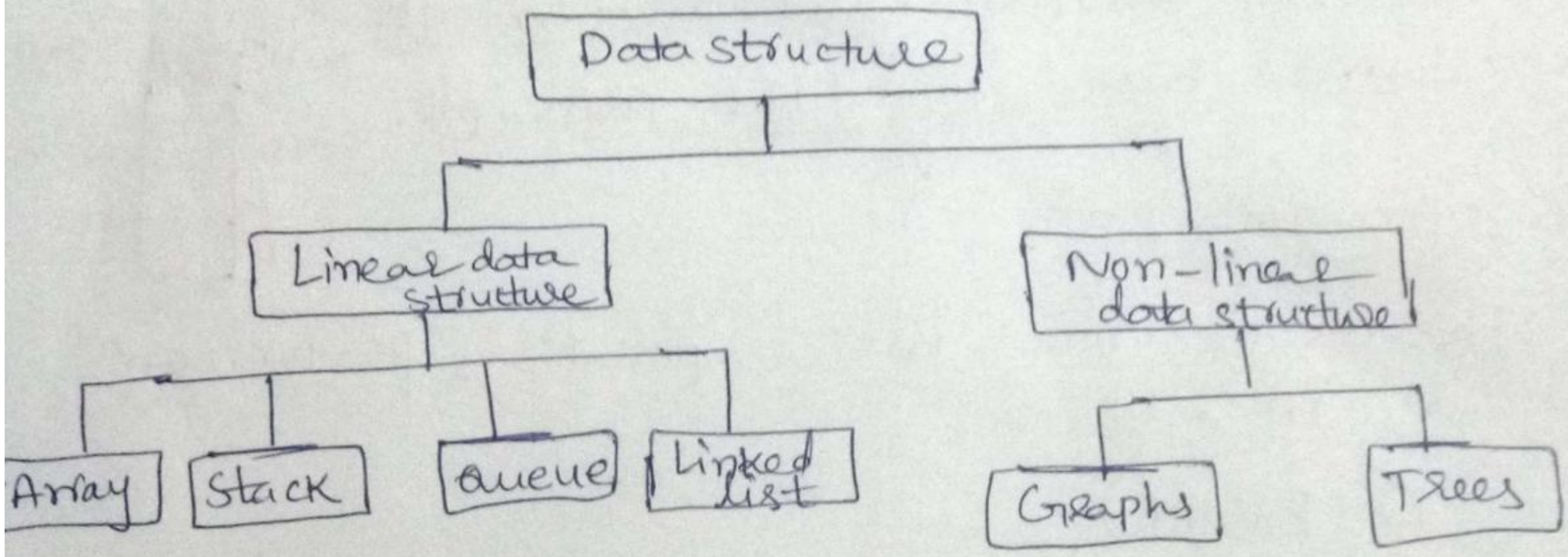
→ A data structure is a collection of data values, the relationship among them, and the functions (or) operations that can be applied to the data.

Types of Data structure :—

There are 2 types of data structure

→ Linear data structure

→ Non - linear data structure



Difference between Linear and Non - linear data struc

Linear data structure :—

Data structure where data elements are arrange sequentially (or) linearly where the elements are attached to its previous and next adjacent ele.

Non-linear Data structure :-

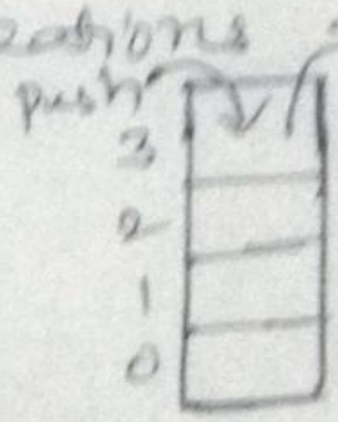
Data structures where data elements are not arranged sequentially (or) linearly are called non-linear data structures.

Stack Data structure :-

→ Stack is a linear data structure which follows a particular order in which the operations are performed. Stack follows the LIFO principle. i.e Last in first out.

(or)

Stack is a collection of similar data items in which both Insertion and deletion operations are performed based on LIFO principle.



Stack operations :-

There are two basic operations performed in a stack.

1) push()
2) pop()

1. push() :- This function is used to add (or) ~~into~~ insert new element into the stack.

. pop() :- This function is used to delete (or) remove an element from the stack.

→ when a stack is completely full, it is said to be overflow state. and if stack is completely empty it is said to be underflow state.

→ stack allows opeeatoins at one end only.
we can only acess the top element of a stack

→ According to its LIFO structure, the element which is inserted last is accessed first.

Implementation of stack :-

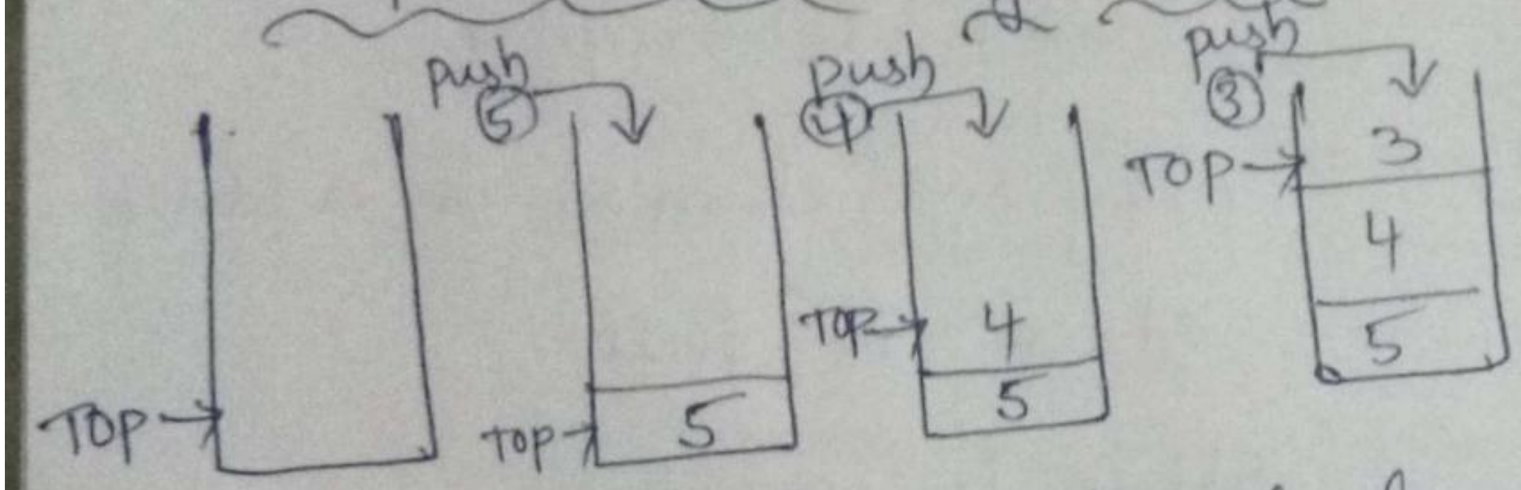


Fig: Insertion of elements in a stack.

→ The above diagram represents a stack insertion opeeation.

→ In a stack, inserting and deleting of elements are performed at a single position which is known as TOP.

→ Insertion operation can be performed using Push() function. New element is added at top of the stack and removed from top of the stack.

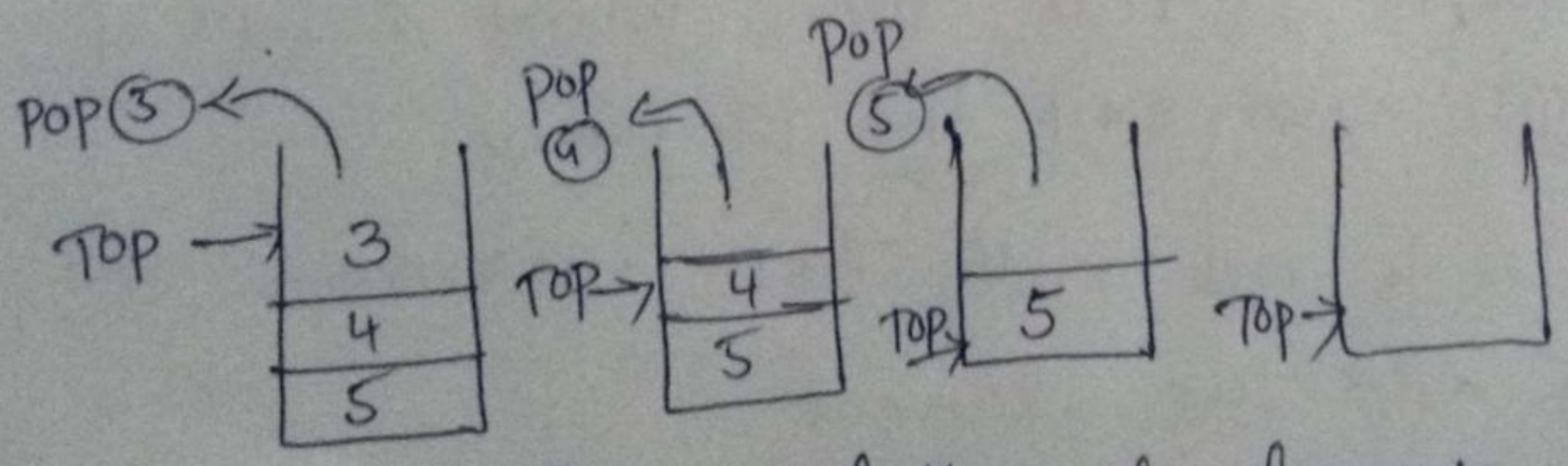


Fig: Deletion of elements in a stack

→ An element is removed from from top of the stack. Delete operation is based on LIFO principle This operation is performed using a POP() function. It means that the insertion and deletion operations are performed at one end i.e TOP.

Position of TOP :-

Position of TOP	Status of stack
-1	Stack is empty
0	only one element in a stack
N-1	Stack is full.
N	Stack is overflow.

Applications of stack :-

① Parsing

2) Expression conversion (infix to postfix, Postfix to prefix etc)

Algorithm for push operation :-

1. check if the stack is full (or) not.
2. If the stack is full, then print error of overflow and exit the program
3. if the stack is not full, then increment the top and add the element.

Algorithm for POP operation :-

1. check if the stack is empty (or) not.
2. if the stack is empty, then print error of underflow and exit the program.
3. If the stack is not empty, then print the element at the top and decrement the top.

Program to implement a stack using Array

```
#include <stdio.h>
#include <stdlib.h>
#define size 10
    int s[size], top=-1, x;
void main()
  {
    int ch;
int s[size], top=-1, ch;
void push(), pop(), display()
do
{
printf(" 1. Insert\n 2. Delete\n 3. Display\n 4. Exit\n");
printf(" enter your choice:\n");
  scanf("%d", &ch);
   switch (ch)
     {
case 1:
        push();
      break;
case 2:
        POP();
      break;
case 3:
        display();
```

```
case 4 :
        exit (0);
        break;
default :
    printf(" Invalid entry. please try again \n");
}
} while (ch! =4);
}
```

```
void insert(ch)
{
    void push()
    {
if ( top == size-1)
    Printf(" stack is full \n");
    else
    {
    Printf(" Enter element \n");
    scanf (" %d ", &x);
      top++;
   s[top] = x;
printf(" element inserted = %d ", x);
}
}.
```

```
void POP()
{
    if (top == -1)
    Printf(" stack is empty \n");
else
  {
        x = s[top];
          top--;
```

```
        printf("element deleted = %d\n", x);
    }
}
void display()
    {
    int i;
  if( top == -1)
  printf("stack is empty\n");
    else
    {
      for( i = top ; i>=0 ; i--)
        printf("%d\n", S[i]);
      }
    }
```

O/P: 1. Insert
2. Delete
3. Display
4. Exit.

enter your choice : 1

enter element : 30

element inserted = 30

1. insert
2. delek
3. display
4. exit

enter your choice : 2

element deleted = 30

enter your choice 3

stack is empty

Queue Data structure

→ A queue is a linear data structure which follows a particular order in which the operations are performed.

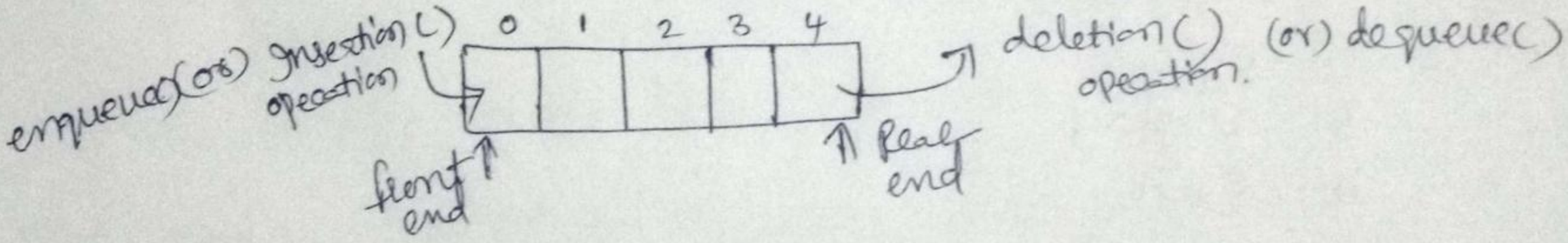
→ queue follows the FIFO principle i,e

first in first out.

(or)

→ Unlike stacks, a queue is open at both its ends. one end is always used to insert data and the other is used to remove (delete) data.

→ Queue follows first-in-first-out order. i,e the data item stored first will be accessed first.



→ In a queue, we add any item at the rear of the queue and remove any item from the front of the queue.

→ Insertions occur at the rear and deletions o the front of the queue. i,e first in first out.

Operations on queue :—

- Insertion() (or) enqueue() :—

This function is used to add (or) insert a item to the queue.

- delete() (or) dequeue() :-

This function is used to delete (or) remove an item from the queue. if the queue is empty, then it is said to be an underflow condition

if the queue is full, then it is said to be an overflow condition.

Applications of queue :—

① when a resource is shared among multiple consumers. Ex: include cpu scheduling, Disk sh scheduling.

2) when data is transferred asynchronously between two processes. Ex: I/O Buffers, pipes, file I/O etc.

Enqueue operation :—

The following steps should be taken to enqueue (insert) data into a queue -

Step1 : check if the queue is full (or) not.

Step2 : if the queue is full, produce overflow error and exit.

Step3 : If the queue is not full, increment rear value. and data element to the queue location.

Dequeue operation :—

The following steps are taken to perform dequeue operation -

Step 1: check if the queue is to empty (or) not

Step 2: if the queue is empty, produce underflow error and exit.

Step 3: if the queue is not empty, access the data where front is pointing

Step 4: increment front pointer to the next elemen

Eg: To implement a queue, we need one array and two variables i,e F(front) & R(Rear)

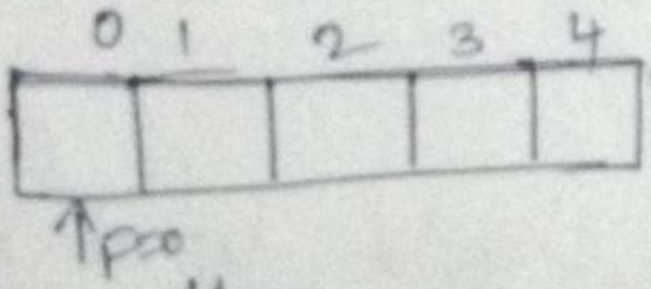
Front(F) → Points to the first element in the queue

(F ↓ deletion)

Rear(R) → Points to the last element in the queue.

(R ↓ Insertion)

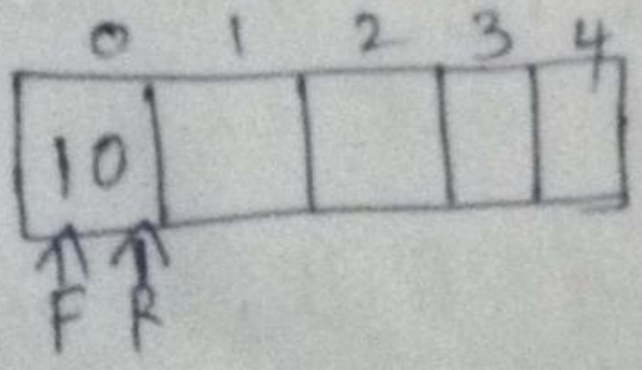
Step 1:—



F=0, R=-1

Initially, the values of F and R are 0 & -1, when the queue is empty

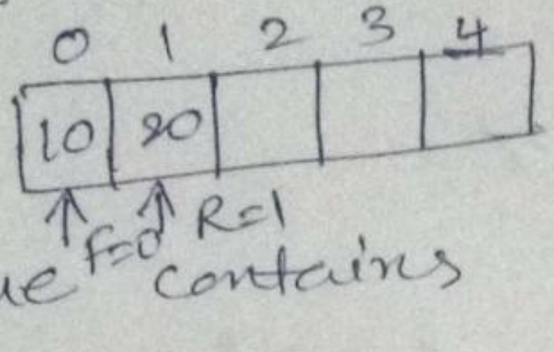
Step 2:— Insert 10 element into the queue.



F=0, R=0

queue contains one elements when F=0, R

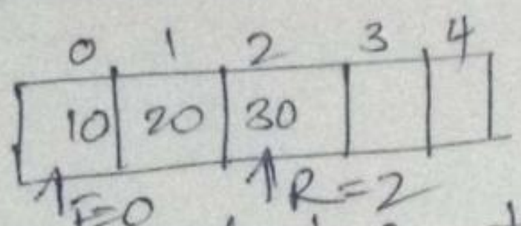
Step 3:- Insert 20 element into the queue



$F=0, R=1$

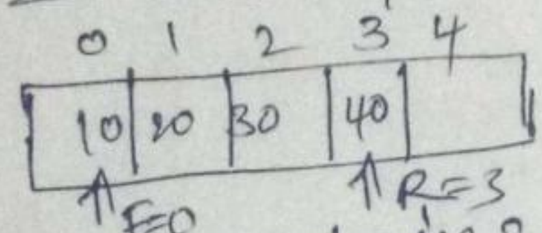
queue contains two elements when $F=0, R=1$

Step 4: Insert 30 element into the queue



queue contains three elements, when $f=0$ & $R=2$

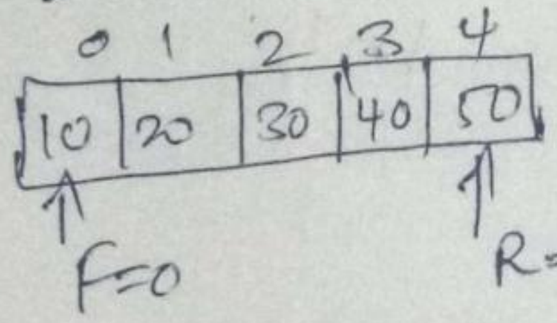
Step 5: Insert 40 element into the queue



$F=0, R=3$

queue contains four elements, when $f=0$ & $R=3$

Step 6: Insert 50 element into the queue



$F=0, R=4,$

size of the queue = 5.

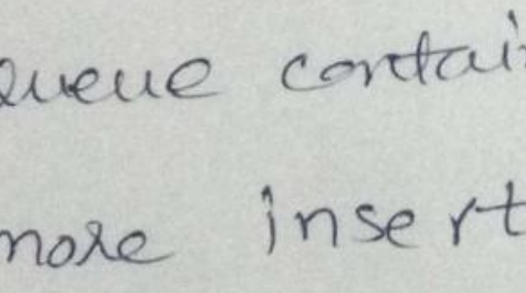
queue contains five elements, when $f=0$ & $R=4$

one more insertion is not possible since

R = SIze - 1; (queue is full).

Dequeue:-

Step 1: Delete First element i.e 10 from the queue after deleting the values of F & R



$F=1$ & $f=4$.

Step 2: Repeat this process until all the elements get deleted from the queue.

* $F > R$ → queue is empty
* $F == R$ → queue contains only one element
* $F < R$ → queue contains more than one element

Program to implement a queue using Array

```
# include <stdio.h>
# include <stdlib.h>
# define size 5
  int a[size], r=-1, f=0;x;
  Void main()
  {
   int ch;
  void insert(), delete(), display();
do
{
printf(" 1. Inser \n  2. Delete \n  3. Display \n
                    4. Exit \n");
Printf(" enter your choice \n");
scanf(" %d", &ch);
 switch(ch)
   {
    case 1:
            insert();
            break;
        case 2:
            delete();
            break;
    case 3:
            display()
             break;
   case 4:
            exit(0)
             break;
```

```
default:
        printf("wrong choice\n");
    }
} while(ch<4);
} // end of main
void insert()
{
if(r==size-1)
  printf("queue is full insertion not possible\n");
else
  { printf("enter element\n");
    scanf("%d",&x);
    r=r+1;
    q[r]=x;
  }
}
void delete()
    {
    if(f>r)
    printf("queue is empty deletion is not
                                possible\n");

else
  {
  printf("element deleted from queue is =%d\n",
                                q[f]));
    f=f+1;
  }
}
void display()
  { int i;
```

```
if( f==0)
  printf(" Queue is empty \n");
  else
  {
    printf(" queue is : \n");
      for( i=f ; i<=r ; i++)
        printf(" %d\t", a[i]);
    }
  }.
```

o/p:

```
1. insert
2. delete
3. display
4. exit
enter your choice
1
enter element
10
enter your choice
1
enter element
20
enter your choice
1
enter element
30
enter your choice
3
10    20    30
enter your choice
2
element deleted from queue =10
enter your choice
2
element deleted from queue =20
enter your choice
2
element deleted from queue =3
```