Date: / / Page no:

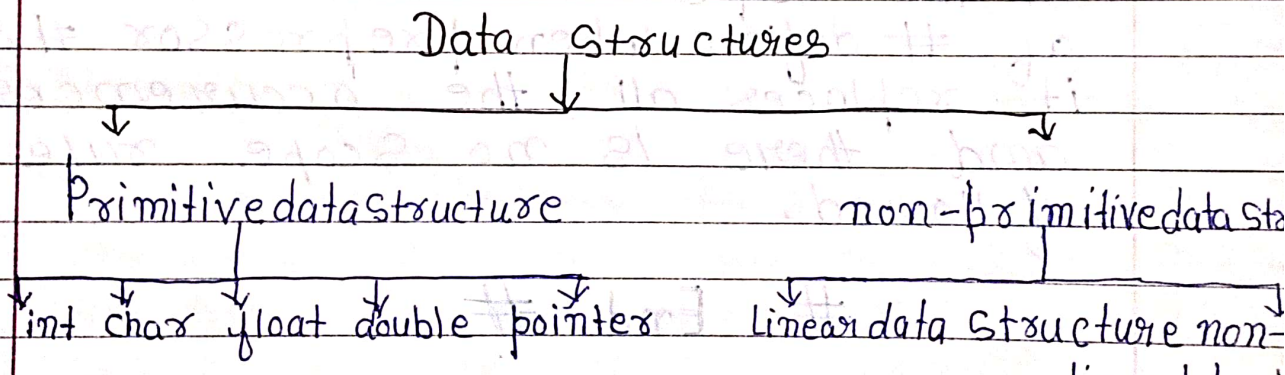
• DSA • (C++)

Data Structure, algorithms :-

Data Structure :- Data structure is a way to store and organize data so that it can be used efficiently.

As per name indicates itself that organizing the data in memory programming languge lik c, c++, java etc.

it is set of algorithm that we can use in any programing languge to structure data in memory.



Linear data structure :- the arrangement of data in the sequential manner is know as linear data structure.

The data structure used for this purpose are arrays linked list stacks and queues.

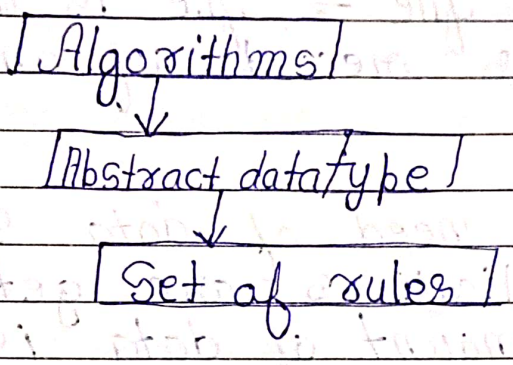
In this data structures, one element is connected to only one another element in a linear from.

Date: / / Page no:

Non-linear data structure :- When on element is connected to the 'n' number of element know as non-linear structure.

ex :- Tree and Graphs

In this case elements are arranged in a random manner.



To structures the data in memory 'n' number, of algorithms are proposed and all these algorithms are know as abstract Data types.

An abstract data typ what is to be done and data structures teles how is to be done.

ADT gives us the blueprint while data structure provides the implementation part.

What is data :- Data can be defined as the elementary value / collection of values.

ex :- Students name and its id are the data about student.

Date: / / Page no:

What is Record ÷ Record can be defined as collection of various data items.
ex ÷ Student entity → name, address, Course, and marks can be grouped together to from record.

What is file ÷ file is a collection of various records of one type of entity.

What is need of data structures?
Ans ÷ As applications are getting complexed and amount of data is increasing day by day there may arrise following problems.

processor speed ÷ as data is growing day by day to the billions of files per entity processor may fail to deal with that amount of data.

Data structure ÷ Consider an inventory size of 100 items in store if our application needs to search for a particular item it needs to transverse 106 items every time results in Slowing down power.

Date: / / Page no:

multiple requests :- if thousands of users are searching data simultaneously on a web server, then there are chances that to be failed to search during that process.

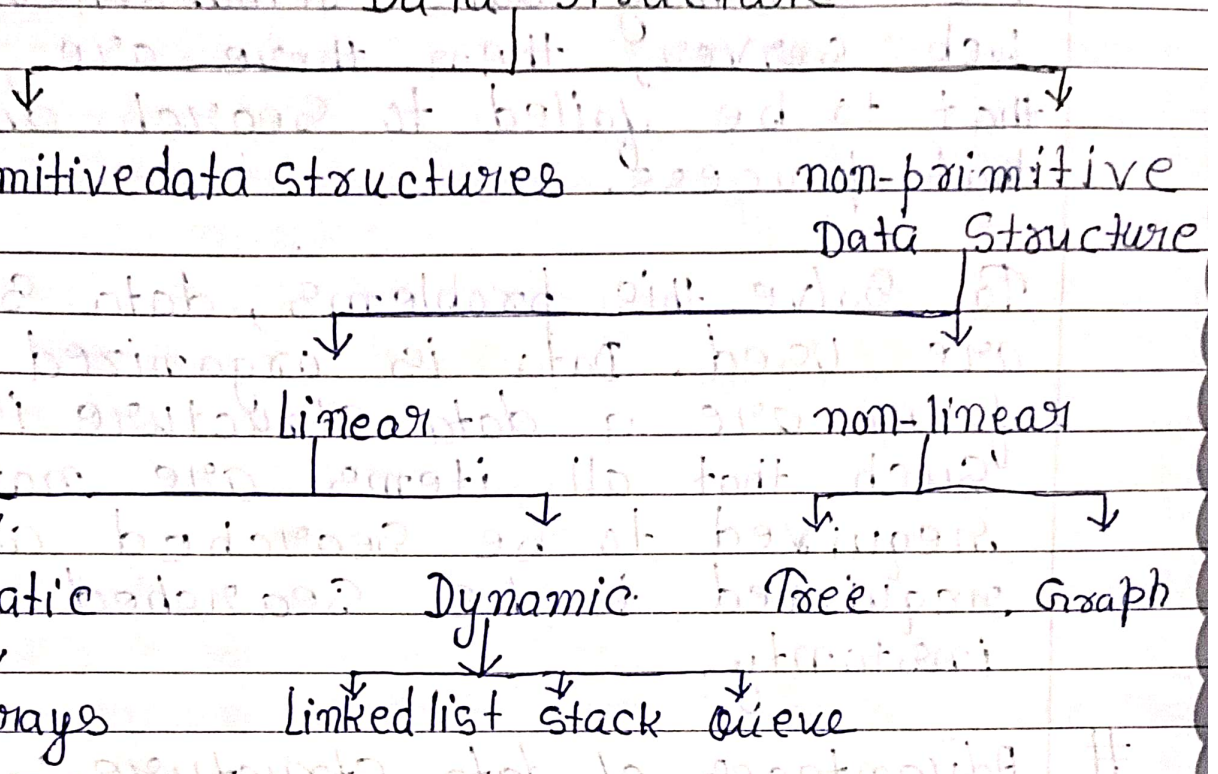
To solve this problems, data structures are used. Data is organized to form are a data structure in a such that all items are note required to be searched and note required to be searched can be instanty.

Advantages of data structure :-

- Efficiency :- If the choice of a data structure for implementing a particular ADT is properit makes program very efficient in terms and space.
- Reusability :- The data structure provides reusability means that multible client programs can be use the data structure.
- Abstraction :- The data structure specified by the ADT also provides level of abstraction. The client cannot see in terval working of data structure so it does not have worry. impleme-ntation.

Date: / / Page no:

Classification of Data Structures



Operations on data structure :-

- Traversing :- Every data structure contains a set of data elements. Traversing data structure, means visiting each element of data structure in order to perform some specific operation like searching or sorting.

- Insertion :- Insertion can be defined as the process of adding the elements to the data structure at any location. If the size of data structures is then be we can only insert n-1 data element to its.

- Deletion :- The process of removing an element from the data structure is called deletion. We can delete an element from data structure at any random location.

If we try to delete an element from an empty data structure then Underflow occure.

- Searching :- The process of finding the location of any element with data structure is called Searching there are two algorithms to perform Searching, linear Search and Binary search.

- Sorting :- The process of arranging the data structure in a specific order called as Sorting.

What is algorithm :- An algorithm is a process or a set of rules required to perform calculation or some other problem Solving operations especially by a Computer.

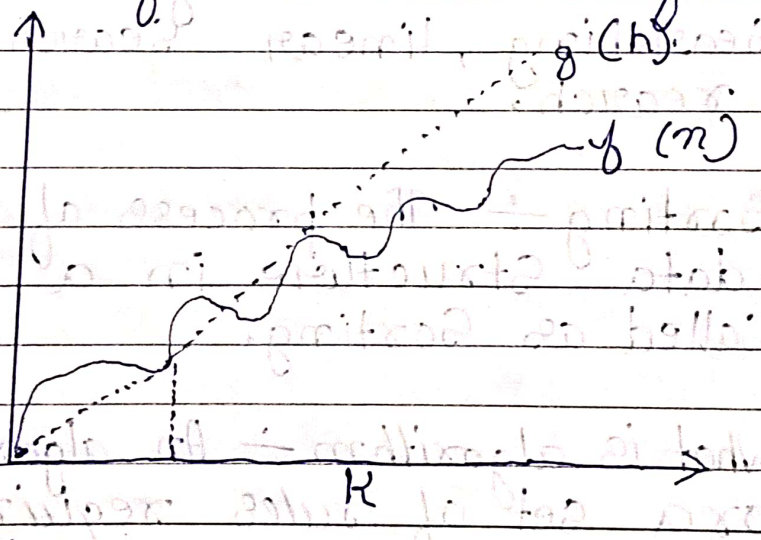
It is not complete program or code it is just a logic of problem an informal description using a flowchart.

Date: / / Page no:

characteristics of an algorithm Input: an algorithm has some input values we can pass some input values to an algorithms.

- Asymptotic notation ÷ The commonly used for calculating the runing time complexity of an algorithm is given below.

- Big oh notation ÷ The measures the performance of an algorithm is given below by simbly providing the order of growth of the notation function.



if F(n) and g(n) are two funcations defined for positive integer then $f(n) = O\ g(n)$ as f(n) is big oh of g(n).

Date: / / Page no:

Array program (1)

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i, a[10];
cout<<"enter the elements in a arrays\n";
for(i=0; i<10; i++)
cin>>a[i];
clrscr();
for(i=0; i<10; i++)
cout<<"a["<<i<<"]"<<a[i]<<endl;
getch();
}
```

Array program (2)

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i, a[10], b[10], c[10];
clrscr();
cout<<"enter the elements in a array A\n";
for(i=0; i<10; i++)
cin>>a[i];
cout<<"enter the elements in a array B\n";
for(i=0; i<10; i++)
cin>>b[i];
for(i=0; i<10; i++) [i]=a[i]+b[i];
clrscr();
cout<<"Sum of two arrays\n";
for(i=0; i<10; i++)
cout<<"c["<<i<<"]="<<c[i]<<endl;
getch();
```

Program

```
#include<iostream.h>
#include<process.h>
#include<conio.h>
#define size 10
class dequeve{
void insert_at_beg(int);
void insert_at_end(int);
void delete_fr_front();
void delete_fr_rear();
void show();
};
dequeve::dequeve(){
f=1;
r=1;
}
void dequeve::insert_at_end(int i){
if(r>=size-1){
cout<<"\n insertion is not possible, overflow!!!!";
}
else{
if(f==-1){
f++;
r++;
}else{
r=r+1
}
a[r]=i;
cout<<"\n inserted item is"<<a[r];
}
}
```

Date: / / Page no:

```
void dequeve:: insert_at_beg (int i)
{
if(f==-1).
{
f=0;
a[++r)=i;
Cout<<"\n inserted element is"<<i;
3
else if (r!= size-1){
{int k=r;
for (int j=k; j>=0; j--)
{a[k+1] = a[k]; 3
r++;
a[f]=1;
cout<<\n inserted element is:"<<j;
3
3
3
void dequeue:: delete_rear_front(){
if (f==-1){
Cout<<"deletion is no possible:: dequeue is
      empty";
return;
3
else{
cout<<" the deleted element is:"<<a[f];
if (f==r){
f=r=-1;
return;
3 else
f=f+1;
```

}
}
void dequeue::delete_for_rear(){
if(f==-1){

• Program (5) •

```
#include<iostream.h>
#include<conio.h>
Void main()
{
int i, a[10];
cout<<"enter the elements in array\n";
for(i=0; i<10; i++)
cin>>a[i];
clrscr();
cout<<"output of array\n";
for(i=0; i<10; i++)
cout<<a[i]<<endl;
getch();
}
```

Write a program add all the elements of array and display the sum.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i, a[10], Sum=0;
cout<<"enter the elements in array\n";
for(i=0; i<10; i++)
cin>>a[i];
clrscr();
```

```
// cout << "output of array \n";
for (i=0; i<10; i++)
 Sum = Sum + a[i];
Cout << " Sum a " << Sum;
 getch();
 3
```

Sum of two arrays (Program)

```
#include <iostream.h>
#include <conio.h>
Void main()
{
int i; a[10] b[10] c[10];
Cout << " enter the elements in array one\n";
for (i=0; i<10; i++)
cin >> b[i];
for (i=0; i<10; i++)
c[i] = a[i] + b[i];
cout << " addition of arrays\n";
for (i=0; i<10; i++)
 Cout << c[i] << endl;
getch();
 3
```

linear searching of numbers in an array.

```
#include <iostream.h>
#include <conio.h>
void main()
{
int i, a[10]
Cout << " enter the elements in array one\n";
for (i=0; i<10; i++)
```

```
cin >> a[i];
clrscr();
cout << " enter an elements to be Search \n";
cin >> n;
for (i=0; i<10; i++)
 if (a[i] = n)
{
cout << " numbers found"; break;
3;
if (i=10)
 cout << " number is not found";
 getch();
3
```

Binary Search :-

Step 1

2	5	9	11	20	41	56	87
A[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

$x = 56$

$$mid = \frac{l+x}{2} = \frac{0+7}{2} = 3.5 = 3$$

→ User always give input in ascending order

$x > mid$

$56 > 11$

Binary Search mid

2	5	9	11	20	41	56	87
A[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

L ↓ H

$$M = \frac{l+1}{2} = \frac{0+7}{2} = 3.5 = [3]$$ A[3] = 11

Date: / / Page no:

- User always input in accending order

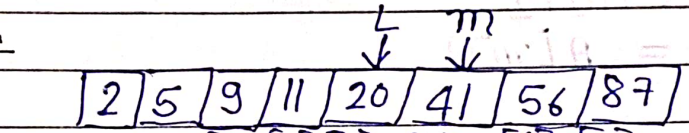
$x > mid$

$56 > 11$

$l = m+1$

$= 3+1 = 4$

- Step 2 :-



$x = 56$

$m = \frac{4+7}{2} = \frac{11}{2} = 5$

$A[5] = 41$

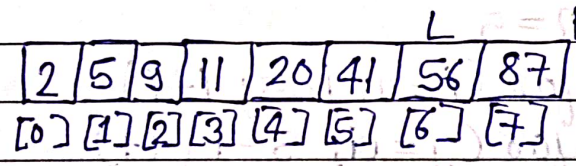
$x > A(M)$

$l = m+1$

$= 5+1 = 6$

$=$ else

- Step 3 :-



$m = \frac{l+H}{2} = \frac{6+7}{2} = 6$

$x == A(m)$

$x == 56$

exit 10

Date: / / Page no:

most important VVI

- Algorithm

```
While (L <= H)
  m = (L+H)/2
  if (x == A[m])
{
cout << "formed";
   exit()
  3
  else if (x > A[m])
    L = m+1
    else
    H = m-1
```

V.V.I

- Bubble Short

```
        n = 5
        for (i = 0; i < 5; i++)
        for (j = 0; 5-i; j++)
  A[0] = 10  if [a])) = a[j+1
  A[1] = 0
  A[2] 2
  A[3] 9
  A[J);
  A[J] = A[J+i);
  A[J+i] = t]
```

Date: / / Page no:

Program :-

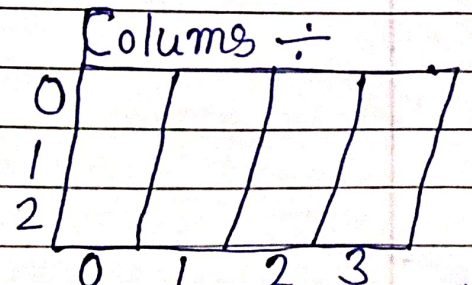
```
#include <iostream.h>
#include< conio.h>
void main()
{
int i, j, a[5]; t i
Cout << " enter elements \n";
for(i=0; i<5; i++)
 for(j=0; j<5-i; j++)
  if (j] > a[j] i]
{
 t= A[j];
 A[j]= A[j+1];
 A[j+1]= l;
 3
 for(i=0; i<5; i++)
 Cout << 0[i]<< endl;
 getch();
 3
```

2-D array :- A 2-D array is a collection of elements placed in m-rows and n Colums Syntax use to declare a 2-D array include 2- Subscript of which one Specified the no of rows and two other Specified the no of - colums of an arrays.

ex: A[3][4]

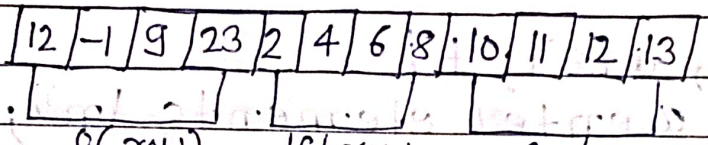
m n

Colums :-



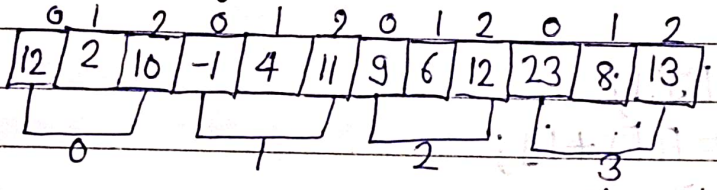
Date: / / Page no:

```
Int A[3][4] = {
              {12, -1, 9, 23 3,
              {2, 4, 6, 8 3,
              {10, 11, 12, [13},
              3;
```



Row major

Row major arrangement :- Element 8 is presentat



Colum major arrangement

a[1][3] = 8

i j

Base address = 502

formula = base add + i x m + j

= 502 + 1 x 4 + 3

Colum major: baseadd + j x m + i

= 502 + 3 x 3 + 1

```
#include<iostream.h>
#include<conio.h>
  void main()
{
int i,j, a[3][3];
cout<<"Enter elements for a matrix\n";
for(i=0; i<3; i++)
for(j=0; j<3; j++)
cin>>a[i][j];
cout<<"matrix\n";
for(i=0; i<3; i++)
{
   for(j=0; j<3; j++)
   cout<<a[i][j]<<"  ";
   cout<<"\n";
   3
   getch();
   3
```

Stack :- A stack is a collection of homogene any one type of elements.
ex:- int, float, char, etc.

LIFO :- Last in first out

4	62	top=4
3	46	top=3
2	42	top=3
1	32	top=1
0	20	top=0

```
max=5;
Top==-1;
if(Top==max-1)
{
cout<<"Stack is full";
return;
3
```

```
else
  top ++;
  A[top] = P;

Max 5;
Top == -1;
if (Top == -1)
{
  cout << " Stack is empty";
   return Null;
  }
  else
   {
   t = A [top];
     top --;
     return (t);
     }
```

→ pop operation

- In c++ class is a user defined data type
→ Access Specifiers : private
Public
Protected

→ Class का data अगर हम Access Specifier नहीं देते हैं तो ओ by default private ho jata hai

→ Constructer is a Special member function of class which has same name as class and it does not have any return types.

Program :-

:: - Scope resolution operator

```
Const  max = 5;
class  stack
{
   private : int a [max], top;
   public : stack;
   void push (int p);
   int pop ();
   };
```

Program :-

```
#include<iostream.h>
#include< conio.h>
  class stack
  {
  private: int a [5];
           int top;
  Public :
      stack ()
    {top = - 1; }
    void push (int p);
     int pop ();
    };
      void stack :: push (int p)
      {
       iF (top == 4)
       {
        Cout<<" stack is full";
        return;
       }
       else
       {
```

```
    top ++;
    a[top] = p;
    }
}
int Stack :: pop ()
    {
    IF (top == -1)
    return Null;
    else
    {
    int k;
    k = a [top];
    top --;
    return(k);
    }
    }
void main ()
{
Stack S1;
S1. push (10);
S1. push (5);
S1. push (8);
S1. pop ();
S1. pop ();
getch ();
}
```

Date: / / Page no:

Application of Stack :-

Note :- operator precedence : (), ↑, /, *, +, -

Algorithm :- Suppose Q is an arthimatic expression written in - fixed notation Q = (a+b*c), this algorithm finds the equivalent post fix expression push C onto the stack and add.

Step 2 :- Scan Q from left to right and repeat Step 3 to 6 for each element of Q Until the Stack is empty.

Step 3 :- If an operand is encountered add into P

Step 4 :- If a left paranthesis C is encountered push it to the stack.

Step 5 :- If an operator is encountered a Repeat pop from the stack and it top each operator top of the stack. Which has Some precedence or heigher than the operator encountered.
b. Add the encountered operators stack

Step 6 :- If a right paranthesis ')' is encountered @ Repeatedly pop from the stack and add to each operator on the top of the Stack Until left parthes is encountered.

⑥ Remove the left parenthesis [Do not add left paranth top]
end of if structure
End of step to loop
Step 7:- exit.

4 $ 2 * 3 - 3 + 8/4 / (1+1)

Character Scanned	stack contents	posifix expression
4	empty	4
$	$	4
2	$	42
*	*	42$
3	*	42$3
+	-	42$3*
8	+	42$3*3-
/	+	42$3*3-8
4	+/	42$3*3-84
/	+/	42$3*3-84/
C	+/	42$3*3-84/
1	+/C	42$3*3-84/1
+	+/C+	42$3*3-84/1
1	+/C+	42$3*3-84/11
)	+/C)	42$3*3-84/11+/C
(a+b)*	C+d)/4(4*a)/	(2*a)↑2

Date: / / Page no:

Infix to prefix.

① If the character scanned happens to be of op. then that character is skipped.

② If the character scanned is a digit or an albhapetitis aded to target string at a

③ If the character scanned is a closing paras then it added to the stack by calling the push operation.

④ If the character scanned happens to be an operator then first we the top most element from the stack is retrived then through a while loop the priorities of the character poped operators are compared.

Following steps are now performed for the precedence rules.

① If the priorites of operator is higher than the charactered scanned the operator gets added to the target string.

⑪ If operator has lower or the same priorites then the character scanned them.

then character push to the stall.

(iii) If the character Scand happen to be an opening paranthesis then operator. enter to the stack are retrived through a loop. loop continees till if does not encountered by closing paranthes. The operators poped are added to the target string.

Application of Stack
① If is used for postfix or prefix
② Nested loop

Queue :- It is also work as FIfo System

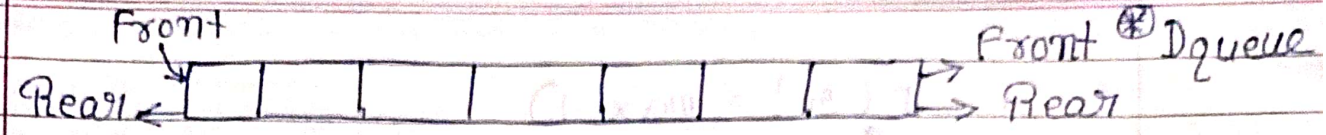
Constructor is a member funcation, the name of Constructor same as class. It does not return values.

Addiang an elements in queue.

```
#include <iostream.h>
#include < conio.h>
#include <process.h>
    class queue
    {
      private : int front, rear;
                int a[6];
      public :
                queue ()
              { front = -1;
                rear = -1;
```

```
void addq(int p);
int delq();
};
void queue:: addq(int p)
    {
      IF(rear==5)
      { cout<<" queue is full";
        exit(0);
      }
    else
      {
       Front=0;
       rear++;
       a[rear]=P;
      }
    }
int queue :: delq()
    {
      int k;
      IF(Front== rear)
      { cout<<" queue is empty";
                exit(0);
      }
      else
        { k=a[Front];
          Front++;
          return(k);
        }
    }
```

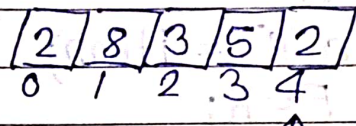
Date: / / Page no:



Dqueue

The word dqueue is a form of double ended queue and defines a data structure in which data can be deleted or inserted other the front or rear end but no changes can be made else where in the list.

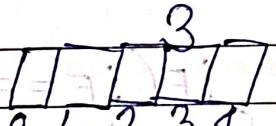
Case-I



max=5

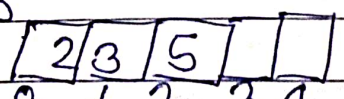
```
if ((F==0) && (R == max-1)
{
    cout<<" queue is full ";
    exit (0);
```

Case II

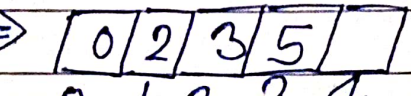



```
if (F==-1) && (r==-1)
{
F=0
r++
a[r] = p;
```

Case III Shifting

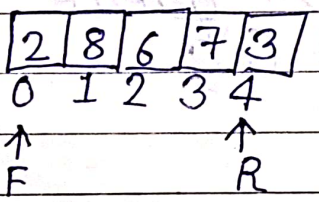


⇒



```
if(r != max-1)
{
  int c = count();
  int k = r+1;
  for(int i = 1; i <= c; i++)
    {
      a[k] = a[k-1];
      k--;
      3;
      a[k] = item;
      f = k;
      r++;
      3
```

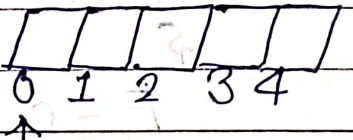
Condition I: Insertion from rear



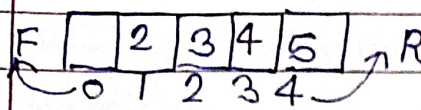
max = 5

```
if(if == 0) && (r == max-1)
{cout << "queue is full"; 3
```

condition II:

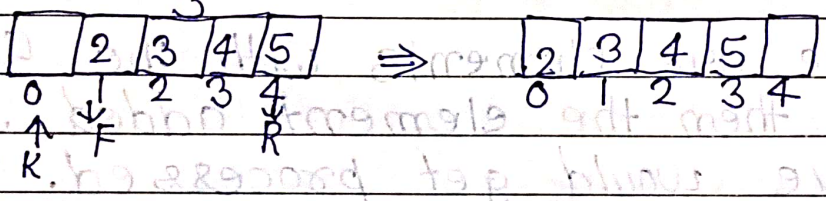



```
if(f == -1) && (r == -1)
{ f == 0;
  i++;
  a[r] = p;
  3
else
  r+f;
  a[r] = p;
```




```
if (r == max-1)
{ int k = -1;
```

- For (i = F-1; i <= r; i++)
  { k = i;
  if (k == max-1)
  a[k] = p;
  else
  a[k] = a[i+1];
  }

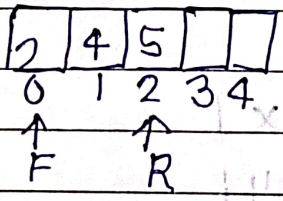


Removing in element from begning.

Condition I:


```
if (if == -1) && (r == -1)
{ cout << " queue is empty";
```

Condition II:




```
else
{ t = a[F];
  F++;
  return (t);
}
```

Deletion from the end of queue

```
if ((F == -1) && (r == -1)
{ cout << " queue is empty
else
{ t = a[r]
```

```
r--;
return (t);
3
```

Priority queue :- A priority queue is a collection of elements where the elements are stored allordingly to their priority level. order in which the element should get added or removed is decided by the priority of an elements.

Following rules are

① The there are elements with the same priority then the element added first is queue would get processed.

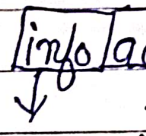
Array implematation in data

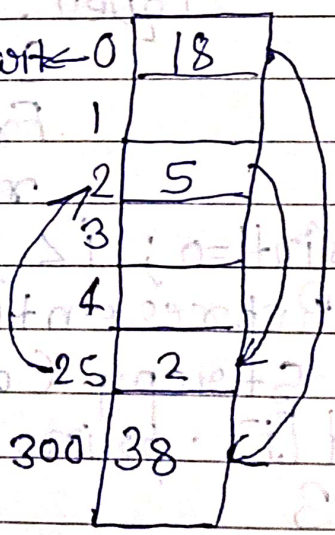
```
Struct data
{
  int a [max];
  int priority;
  int order;
  3;
```

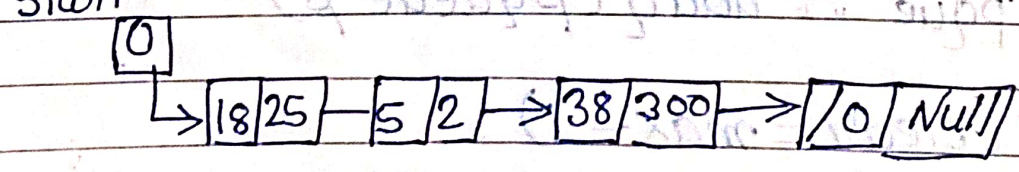
```
#include<iostream.h>
#include<conio.h>
#include<string.h>
    Const int max = 5;
    Class pqueue
    {
    private:
            Struct Pqueue
             { char job(max);
               int prino;
               int order;
             } d(max);
    Public:
     Pque();
     void addq(Pqueue P);
      pqueue delq();
     };
    Pqueu :: pque()
          {
           Front = -1;
           rear = -1;
for(int i=0; i<; i++)
  Representation of Null
   Strcpy(d[i], \0);
 d[i].prino = d[i] order = 0;
 }
 void pque :: addq(pqueue p)
  {
   iF(rear = max-1)
   {
   cout<<"queue is full";
```

```
}
else {
    rear++;
    d[rear] = p;
}
if (front == -1)
{
    front = 0;
}
pqueue temp;
for (int i = front; i <= rear; i++)
{
    if (d[i].prino > d[i+1].prino)
    {
        temp = d[i].prino;
        d[i].prino = d[i+1].prino;
        d[i+1] = temp;
    }
}
```

Linked list

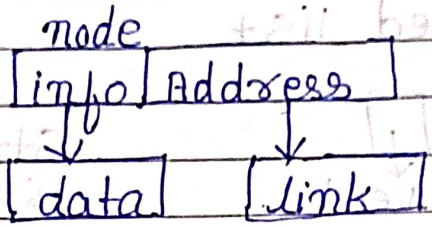




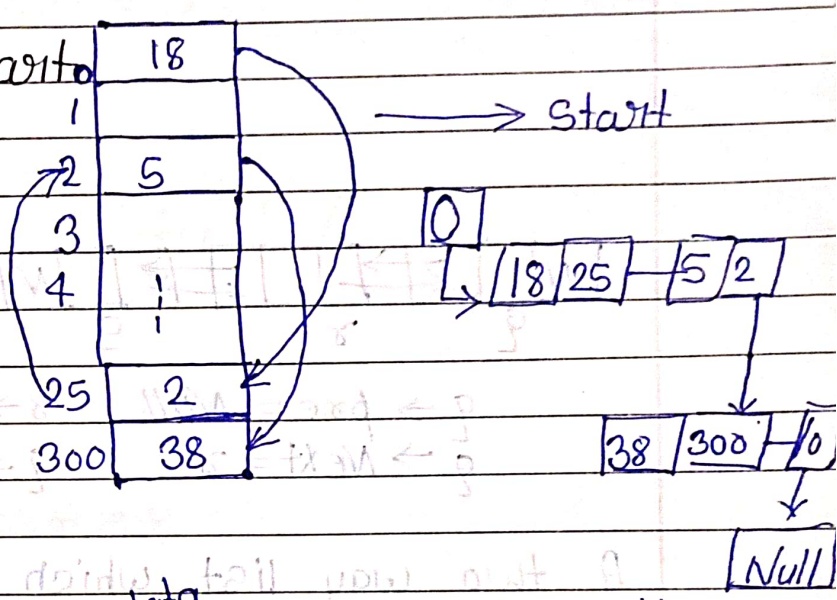


Linked list

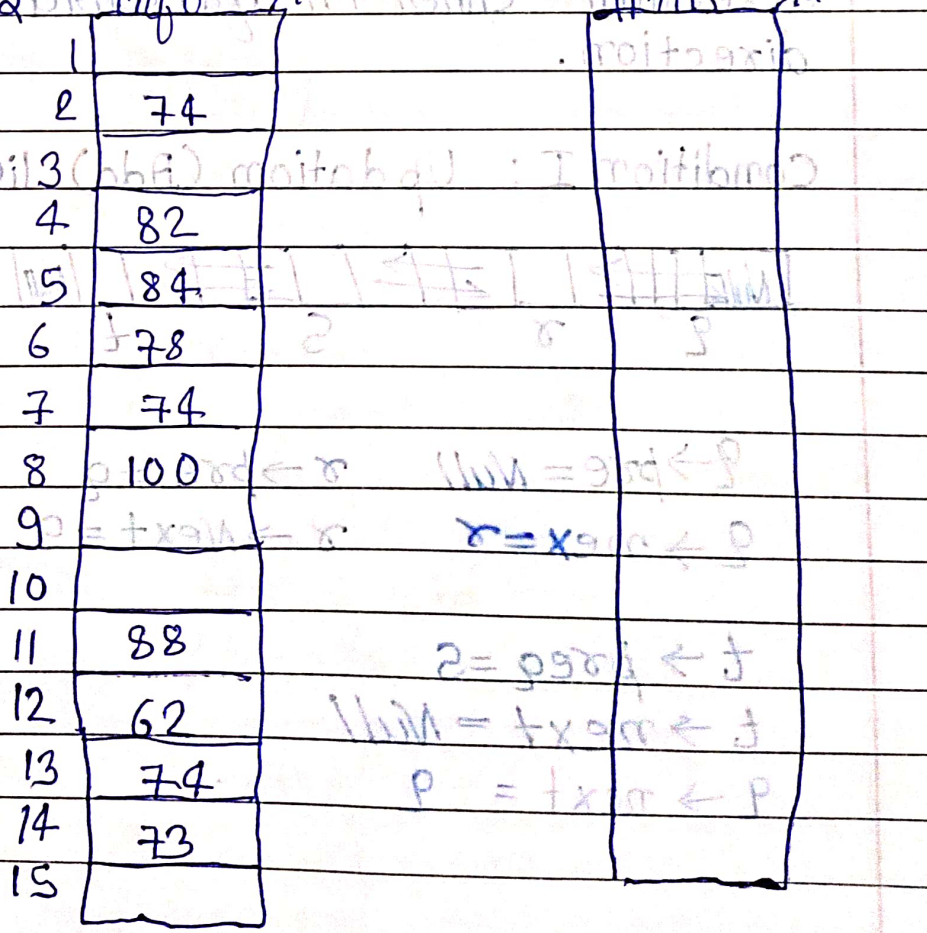
Example :-



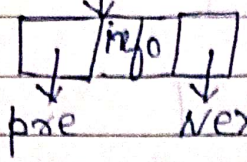
Example :-1



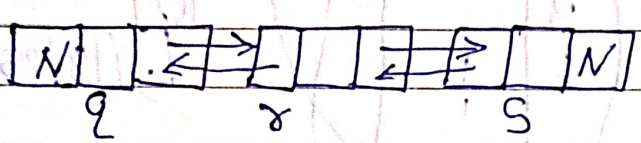
Example-2



Doubly linked list



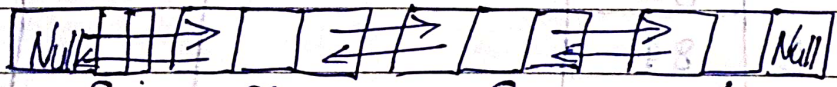

```
Struct dnode
{
int info ;
dnod * pre;
dnod * next;
3 *q;
```



q → pre = Null r → pre = q r → next = S
q → Next = r q → next = r S → pre = r
S → next = Null

A two way list which can travalary in two direction either in forward or backward direction.

Condition I: Updation (Add) list from last

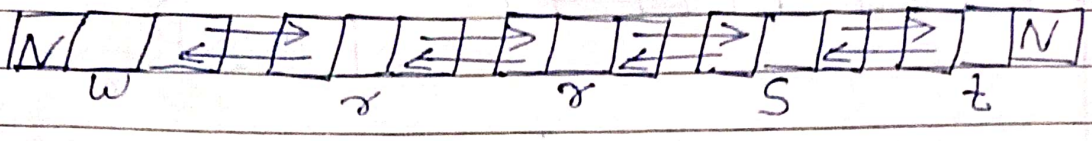


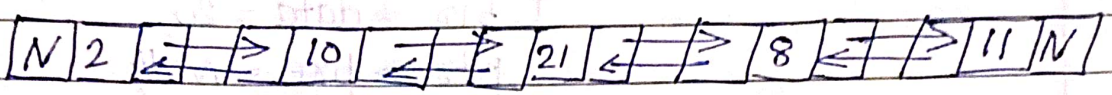
q → pre = Null r → pre = q S → pre = r
q → nex = r r → Next = S S → next = t

t → preq = S
t → next = Null
q → next = q

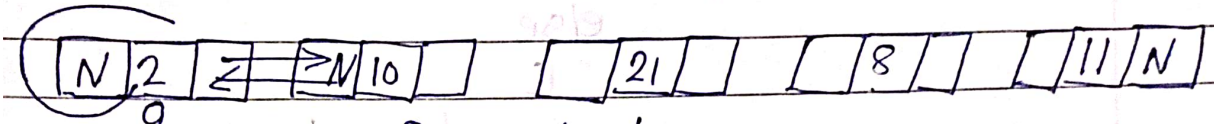
Date: / / Page no:

Condition II: Add list from beginning





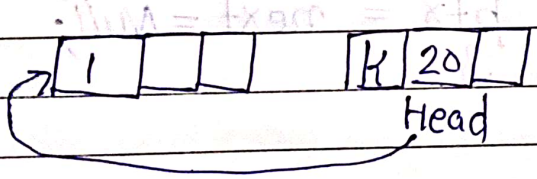
Deletion:



q -> Next

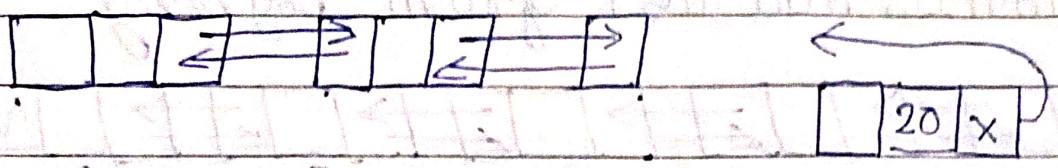
q -> next -> pre = Null
q -> next = Null
q -> next = q

Condition I: Insertion at the beginning of node.




```
{ ptr -> data 20;
  ptr -> pre = Null;
  ptr -> next = Null;
  3
  else
     ptr -> data
     ptr -> pre = Null;
     ptr -> Next = Head;
     Head = ptr;
```

Date: / / Page no:




```
{ ptr -> data = 20;
  ptr -> pre = Null;
  ptr -> next = Null;
}
else
  {
    node * temp;
     temp = head;
  While (temp -> next = Null)
  {
  temp = temp -> next;
  temp = next = ptr;
  ptr = pre = temp;
  ptr = next = Null;
  }
```