Data-structure-

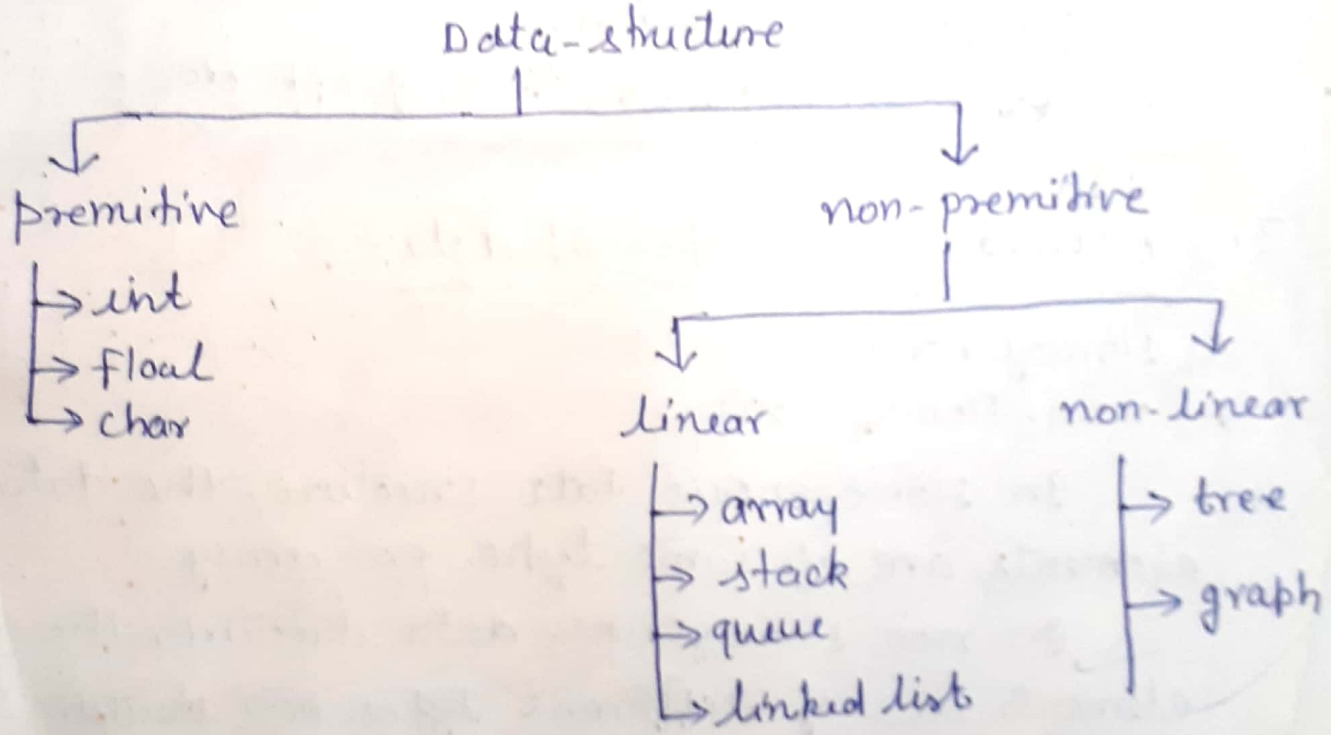
- Data structure is a format for storing and organizing data.
- Data structure is a way of organizing all data items that consists not only to store data items but also their relationship to each other.

Logical Description
Implementation
Quantitative analysis

Need of data structure-

Classification of data structure :-

1) According to occurrence :-



In linear data structure data is stored in contiguous memory location or data is stored in a sequential form, i.e. every element has a unique predecessor and unique successor.

In non linear data structure, data is not arranged in sequence. It is mainly used to represent data

(2) In this, there is no unique predecessor or unique successor.

2) According to nature of size :->

a) static data structure
b) dynamic data structure

A data structure is said to be static, if we can store data upto a fix number.
ex-> array

(b) A data structure is said to be dynamic, if size of the data may change during program execution.
ex-> linked list, tree, graph etc.

3) According to types of data:-

a) Homogenous
b) non-Homogenous

In Homogenous data structure, the data elements are of same type. ex-> array

In non-Homogenous data structure, the data elements are of different types. ex-> structure

Difference between linear and non-linear data structure:

Linear	Non-Linear
elements are arranged in a linear order where each and every element is attached to its previous and next adjacent	data elements are attached in hierarchically manner.
Single level is involved.	multiple levels are involved.
gts implementation is easy.	gts implementation is complex.
data elements can be traversed in a single run	data elements can't be traversed in a single run.
examples are array, stack, queue, linked list etc.	examples are tree and graph.

Algo

Abstract data type

- Abstract data type is a type for object whose behaviour is defined by set of values and set of operations.
- The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented.
- It does not specify how data will be organized in memory.
- It is called abstract because it gives an implementation-independant view.

example→ on a smartphone the specification is 8GB RAM, 5 inch LED screen, Dual camera etc. these specifications are data.

We can perform the following operations:

- call(): We can call
- text(): We can text messages
- photo(): We can click the photo
- video(): We can make video

Algorithm- An algorithm is any well defined computational procedure that takes some value or set of values as input and produces some value or set of values as output.

Charactristics of Algorithm

1. Input:- Zero or more quantities are externally supplied.
2. Output:- At least one quantity is produce.
3. Definiteness:- Each instruction is clear and unambiguous.
4. finiteness:- The algorithm terminates after a finite number of steps.
5. Effectiveness:- in Algorithm, it is not enough that each operation be definite but it must be feasible.

Analysis of Algorithm-

) **Worst case Running time**:- The worst case of analysis is the function which defines the maximum number of steps taken for a data of size N.

) **Best case**:- The best case of an algorithm is the function which defines the minimum number of steps taken for a data of size N.

) **Average case**- The average case of analysis is the function which defines the average number

of steps taken for a data of size N.

example→ Linear_search(A, N)
begin

1. set K:=1 and LOC=0
2. repeat step 3 and 4 while(K<=N)
3. if item= DATA[K] then set loc=K
4. set K=K+1

[END OF ~~STEP~~ LOOP 2]

5. if LOC=0 then
   write " item is not found"
   ELse
   write " item is found"
6. END

Worst case- when the item is in last location or not in an array.

$$C(n) = n$$

Best case:- when the item is in first location.

$$C(n) = 1$$

Average case:- probability of each number occure $= \frac{1}{n}$

$$C(n) = 1.\frac{1}{n} + 2.\frac{1}{n} + 3.\frac{1}{n} + \cdots + n.\frac{1}{n}$$

$$= (1+2+3+\cdots+n).\frac{1}{n}$$

$$= \frac{n(n+1)}{2}.\frac{1}{n} = \frac{n+1}{2}$$

Rate of growth

Asymptotic Notation

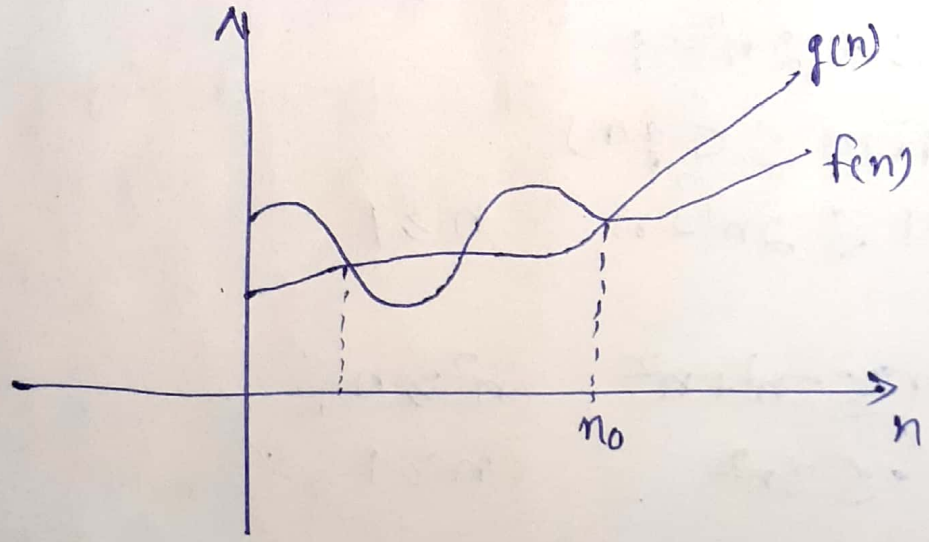
Rate of growth or Big O notation

suppose $f(n)$ and $g(n)$ are functions defined on positive integers with the property that $f(n)$ is bounded by some multiple of $g(n)$ for allmost all n. suppose there exists a positive integer n_0 and a positive number c such that for all $n > n_0$ we have

$$|f(n)| \leq c\,|g(n)|$$

then we may write

$$f(n) = O(g(n))$$



examples → $f(n) = 3n + 1$

we know that for O notation

$$f(n) \leq c\,g(n)$$

so

$$3n+1 \leq 3n+n \quad n \geq 1$$

compare with $f(n) \leq c\, g(n)$

$g(n) = n \quad c = 4 \quad n_0 = 1 \quad f(n) = O(g(n))$

Ex-2→ $f(n) = 4n^2 + n + 1$

$f(n) \leq c\, g(n)$

$4n^2 + n + 1 \leq 4n^2 + n + n \quad n \geq 1$

$\leq 4n^2 + 2n$

again $4n^2 + n + 1 \leq 4n^2 + n^2 \quad n^2 \geq 2n$

$n \geq 2$

$4n^2 + n + 1 \leq 5n^2$

$g(n) = n^2 \quad c = 5 \quad n_0 = 2 \quad f(n) = O(n^2)$

Ex-3- $f(n) = 2n^2 + 1$

$f(n) \leq c\, g(n)$

$2n^2 + 1 \leq 2n^2 + n \quad n \geq 1$

Again

$2n^2 + 1 \leq 2n^2 + n^2 \quad n^2 \geq n$

$\leq 3n^2 \quad n \geq 1$

$g(n) = n^2 \quad c = 3 \quad n_0 = 1 \quad f(n) = O(n^2)$

Ex-4- $f(n) = n^3 + 2n^2 + n + 1$

$f(n) \leq c\, g(n)$

$n^3 + 2n^2 + n + 1 \leq n^3 + 2n^2 + n + n \quad (n \geq 1)$

$\leq n^3 + 2n^2 + 2n$

Again

$n^3 + 2n^2 + n + 1 \leq n^3 + 2n^2 + n^2$ $\quad n^2 \geq 2n$

$\leq n^3 + 3n^2$ $\quad n \geq 2$

Again

$n^3 + 2n^2 + n + 1 \leq n^3 + n^3$ $\quad n^3 \geq 3n^2$

$\leq 2n^3$ $\quad n \geq 3$

$c = 2 \quad g(n) = n^3 \quad n_0 = 3 \quad f(n) = O(n^3)$

5- $f(n) = 4n^3 + 2n + 3$

$f(n) \leq c\, g(n)$

$4n^3 + 2n + 3 \leq 4n^3 + 2n + n$ $\quad n \geq 3$

$\leq 4n^3 + 3n$

Again

$4n^3 + 2n + 3 \leq 4n^3 + n^2$ $\quad n^2 \geq 3n$

$\leq 4n^3 + n^2$ $\quad n \geq 3$

$4n^3 + 2n + 3 \leq 4n^3 + n^3$

$\leq 5n^3$ $\quad n \geq 5$

Ex 6- $f(n) = 2^n + 3n^2 + 2n$

$f(n) \leq c\, g(n)$

$2^n + 3n^2 + 2n \leq 2^n + 3n^2 + n^2$ $\quad n^2 \geq 2n$

$\leq 2^n + 4n^2$ $\quad n \geq 2$

Again

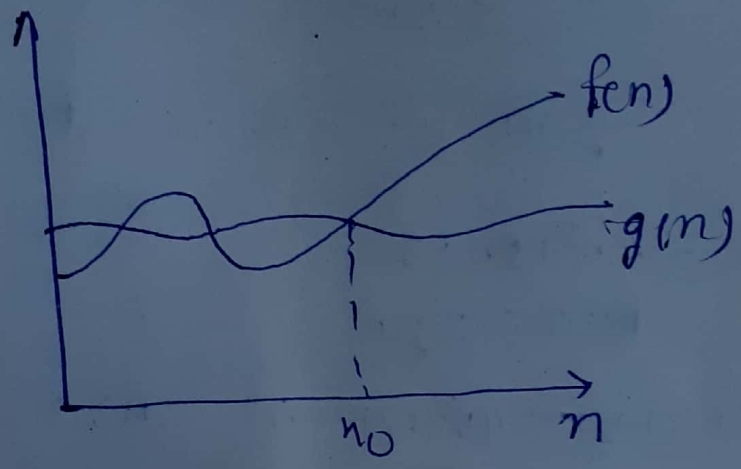
$2^n + 3n^2 + 2n \leq 2^n + 4 \cdot 2^n$ $\quad 4 \cdot 2^n \geq 4n^2$

$\leq 5\, 2^n$ $\quad 2^n \geq n^2 \quad n \geq 4$

$c = 5 \quad g(n) = 2^n \quad n_0 = 4$

2) Omega Notation (Best case) (Ω) (10)

$f(n) = \Omega(g(n))$ iff there exists a positive integer n_0 and a positive number C such that if $|f(n)| \geq C|g(n)|$ for all $n \geq n_0$



example1- $f(n) = 4n + 4$

we know

$$C\,g(n) \leq f(n)$$

$$4n \leq 4n + 4$$

$C = 4$ for all n_0 $f(n) = \Omega(n)$

Ex 2- $f(n) = 2n^2 + 3n + 1$

$$C\,g(n) \leq f(n)$$

$$2n^2 \leq 2n^2 + 3n + 1$$

$C = 2$ for all n_0

$f(n) = \Omega(n^2)$

Ex 3- $f(n) = 7n^3 + 4$

$7n^3 \leq 7n^3 + 4$

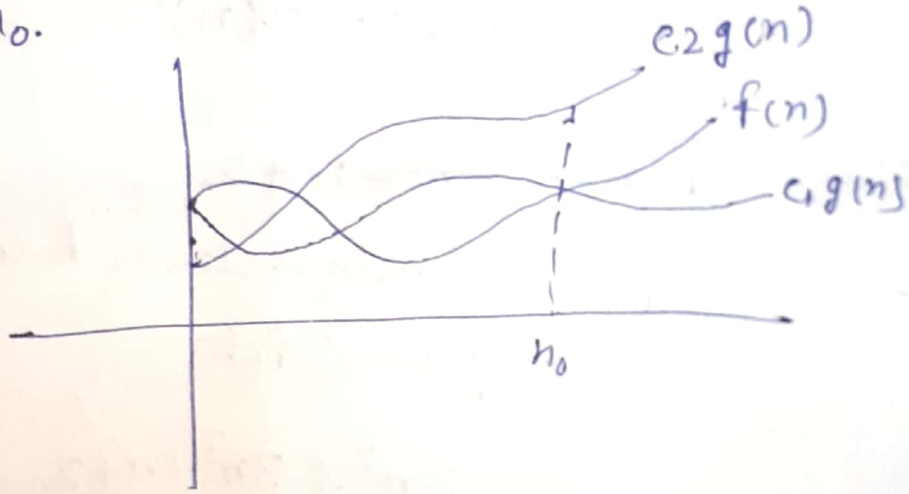
$c = 7$ for all n_0

$f(n) = \Omega(n^3)$

Theta Notation (θ) (Average case)

$f(n) = \theta(g(n))$ iff there exists two positive constant c_1 and c_2, and a positive integer n_0, such that

$c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|$ for all $n \geq n_0$.



Ex 1→ $f(n) = 2n + 7$

We have to find lower bound (Ω) and upper bound (O) for calculating θ.

$c_1 |g(n)| \leq f(n) \leq c_2 |g(n)|$

So first calculate lower bound

$c_1 |g(n)| \leq f(n)$

$2n \leq 2n + 7$

$c_1 = 2$ for all n_0

now calculate upper bound

$$f(n) \leq c_2 g(n)$$

$$2n+7 \leq 2n+n \qquad n \geq 7$$

$$\leq 3n$$

$c_2 = 3$ & $n_0 = 7$

so compare with

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$2n \leq 2n+7 \leq 3n$$

$c_1 = 2$ $c_2 = 3$ $n_0 = 7$.

$$f(n) = \theta(n)$$

Ex2:- $f(n) = 3n^2 + n + 2$.

first calculate lower bound

$$c_1 g(n) \leq f(n)$$

$$3n^2 \leq 3n^2 + n + 2$$

$c_1 = 3$ for all n_0

now calculate upper bound

$$f(n) \leq c_2 g(n)$$

$$3n^2 + n + 2 \leq 3n^2 + n + n \qquad n \geq 2$$

$$\leq 3n^2 + 2n$$

Again

$$3n^2 + n + 2 \leq 3n^2 + n^2 \qquad n^2 \geq 2n$$

$$\leq 4n^2 \qquad n \geq 2$$

$3n^2 \leq 3n^2+n+2 \leq 4n^2$

$c1=3 \quad c2=4 \quad n_0=2$

$f(n) = \Theta(n^2)$

Ex 3- $f(n) = 9n^2+4$

calculate lower bound

$c1\, g(n) \leq f(n)$

$9n^2 \leq 9n^2+4$

$c1=9$ for all n_0

now calculate upper bound

$f(n) \leq c2\, g(n)$

$9n^2+4 \leq 9n^2+n \qquad n \geq 4$

$9n^2+4 \leq 9n^2+n^2 \qquad n^2 \geq n$

$\leq 10n^2 \qquad n \geq 1$

4- $f(n) = 3n^3+n^2+n+2$

lower bound

$3n^3 \leq 3n^3+n^2+n+2$

$c1=3$ for all n_0

upper bound.

$3n^3+n^2+n+2 \leq 3n^3+n^2+n+n \quad n \geq 2$

$\leq 3n^3+n^2+2n$

again

$3n^3+n^2+n+2 \leq 3n^3+n^2+n^2 \quad n^2 \geq 2n$

$n \geq 2$

$\leq 3n^3+2n^2$

$3n^3+n^2+n+2 \leq 3n^3+n^3 \quad n^3 \geq 2n^2$

$n \geq 2$

$\leq 4n^3$

$3n^3 \leq 3n^3+n^2+n+2 \leq 4n^3$

$c_1=3 \quad c_2=4 \quad n_0=2$

$f(n)=\theta(n^3)$

Ex:- $f(n)=2^n+6n^2+3n$

lower bound

$2^n \leq 2^n+6n^2+3n$

$c_1=1$ for all n_0

Upper bound

$2^n+6n^2+3n \leq 2^n+6n^2+n^2 \quad n^2 \geq 3n$

$\leq 2^n+7n^2 \quad n \geq 3$

again

$2^n+6n^2+3n \leq 2^n+7\cdot 2^n \quad 2^n \geq n^2 \quad n \geq 4$

$= 8\cdot 2^n$

$2^n \leq 2^n+6n^2+3n \leq 8\cdot 2^n$

$c_1=1 \quad c_2=8 \quad n_0=4 \quad f(n)=\theta(2^n)$

Time space tradeoff :-

A tradeoff is a situation where one thing increases and another thing decreases. It is a way to solve a problem in:

a) Either in less time and by using more space.

b) In a less space by spending a long time.

Example- 1) compressed or uncompressed data

2) smaller code or loop unrolling (less time)

3) lookup table (more space, less time) or recalculation.

loop unrolling

```
main()
{
  int i;
for(i=1; i<4; i++)
  {
    printf("A");
  }
}
```

```
main()
{
printf("A");
printf("A");
printf("A");
}
```

Recalculation

```
int fib(int N)
{
  if (N<2)
    return N;
  else
    return (fib(N-1)+fib(N-2))
}
```

Time complexity = $O(2^n)$

space = $O(1)$

```
int fib(int N)
{
  int f[30]; i;
  f[0]=0;
  f[1]=1;
  for(i=2; i<N; i++)
  {
    f[i] = f[i-1]+f[i-2];
  }
  return(f[N]);
}
```

Time = $O(N)$

space = $O(N)$

Array

Array is a collection of elements of similar datatype.

or

Array is a collection of contiguous memory locations.

It always has a predefined size and the elements of an array are referenced by means of index/subscript.

Arrays are of following types

1) 1 D array or Linear array
2) 2 D array or matrix
3) multidimensional array.

1) 1 D array or Linear array.

Declaration -

```
datatype arrayname [size];
```

ex→ `int arr[10];`

reading a Linear array (input)

```
for(i=0; i<n; i++)
  scanf("%d", &arr[i]);
```

writing a Linear array.

```
for(i=0; i<n; i++)
  printf("%d", arr[i]);
```

representing 1D array in memory

Let A be a One dimensional array with n elements, the system need to keep track of the address of first elements only. Base(A) called base address of A.

The address of any element of A

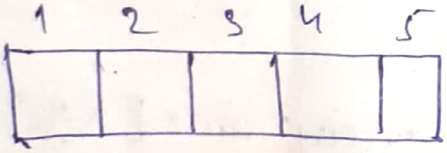
$$LOC(A[K]) = base(A) + W(K - LB)$$

· In c programming $LB = 0$

so $LOC(A[K]) = base(A) + W(K)$

W = no. of bytes per storage location.

Derivation -



Let LB = 1 (address of first element) size = 1 Byte B(A) is the base address

A[1] = B(A)

address of second element

A[2] = B(A) + 1

similarly

A[3] = B(A) + 2

!

A[K] = B[A] + (K-1)

now ~~let~~ if size of each element is w.

A[K] = B[A] + W * (K-1)

A[K] = B[A] + W * (K-LB)

Q:- consider the linear array A[5:50], B[-5, 10] and C[18]

a) find the number of elements in each array

b) Let Base(A) = 300 and w = 4 bytes. Find the address of A[15], A[40] and A[55].

Sol:- No. of elements in array is equal to the length of array.

a) length = UB - LB + 1

length(A) = 50 - 5 + 1 = 46

length(B) = 10 - (-5) + 1 = 16

length(C) = (18 - 1 + 1) = 18

b) Loc[A[15]] = base(A) + w * (K - LB)

= 300 + 4 * (15 - 5)

= 300 + 40 = 340

Loc(A[40]) = 300 + 4 * (40 - 5)

= 440

~~Loc(A[55]) = 300 + 4 * (55 - 5)~~

~~= 500~~

A[55] is not ~~the~~ an element of A. Since UB = 5

Operations on Linear array -

1) **Traversing** - It is the process of visiting each element of an array exactly once, i.e. starting from first element upto the last element.

Algorithm -

Traverse (A, N)
begin.
1. [initialize counter] Set K = LB
2. Repeat step 3 and 4 while K <= UB
3. [visit element] Apply process to LA[K]
4. set K = K+1
   [End of Loop 2]
5. exit

C program -

```
void traverse(int A[], int N)
{
   int K;
   K = 0;      // LB = 0
   while (K < N)
   {
      printf("%d", A[K]);
      K = K+1;
   }
}
void main()
{
```

```
int A[30], i, N;
printf("Enter No. of elements");
scanf("%d", &N);
printf("Enter array elements");
for(i=0; i<N; i++)
    scanf("%d", &A[i]);
    traverse(A, N);
    getch();
}
```

Insertion:-

concept → Let A be a collection of data elements. Insertion refers to the operation of adding another element to the collection of A.

let original array is

A[1]	A[2]	A[3]	A[4]	A[5]
12	13	14	15	16

N = 5 ~~ele~~ = 17 (to be added or inserted)

K = 3 (location at which element to be inserted.

set j to last index

1	2	3	4	5
12	13	14	15	16

move j towards left until reach at K and move all elements at ~~ele~~ ~~no~~ j ~~loc~~ >=K towards right

A[j+1] = A[j]

1	2	3 (K)	4	5	6
12	13		14	15	16

now insert e at loc[K]

by A[K] = e

Now increase the length of an array by

N = N+1

12	13	17	14	15	16

Algorithm –

Insert (A, N, K, ITEM)

Here A is linear array with N elements and K is a positive integer such that K<=N. This algorithm insert an element ITEM into the Kth position in A.

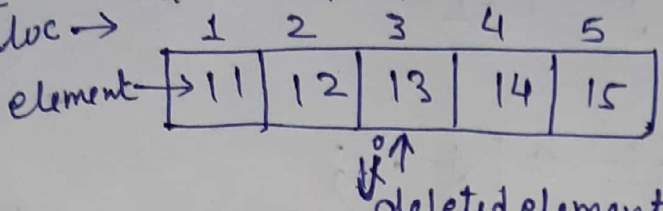
1. Set j = N
2. Repeat step 3 and 4 while (j >= K)
3. A[j+1] = A[j]
4. j = j - 1

   [End of step 2]
5. Set A[K] = ITEM
6. Set N = N+1
7. Exit

```
void insert(int A[], int N, int K, int e)
{
    int j = N-1;
    while (j >= K-1)
    {
        A[j+1] = A[j];
        j = j-1;
    }
    A[K-1] = e;
    N = N+1;
printf("new array is");
  for(i=0; i<N; i++)
    printf("%d", A[i]);

}

void main()
{
  int A[30], i, N, e, K;
 printf("Enter no. of elements");
  scanf("%d", &N);
 printf("Enter array elements");
  for(i=0; i<N; i++)
  {
    scanf("%d", &A[i]);
  }
 printf("Enter element to be inserted");
 scanf("%d", &e);
 printf("Enter location to be inserted");
 scanf("%d", &K);
 insert(A, N, K, e);
}
```

Deletion– Let A be a collection of data elements. Deletion refers to the operations of removing one of the elements from A. Deletion at the end can be easily done. Deleting an element at the middle of a linear array require each subsequent be moved one location upward to fill up the array.

concept→ Let Array A, K.



item

13

Set J = K, item = A[K]

move all element at ~~J to N~~ J+1 to N towards left by

A[J] = A[J+1]

11	12	14	15	

now N = N-1

11	12	14	15

Algorithm→ Delete(A, N, K, ITEM)

Here A is Linear array and K is a positive integer such that K<=N. This element the K^{th} elements A.

1. Item = A[K]
2. Set J = K
3. repeat step 4 while (J<=N-1)
4. A[J] = A[J+1] set J = J+1
   [End of Loop]
5. N = N-1
6. Exit

C program

```
void delete ( int a[], int n, int k)
  {
     int j, e;
      j = k-1;   e = a[k-1];
    while ( j <= n-2)
     {
       a[j] = a[j+1];
         j = j+1;
      }
      n = n-1;
    printf(" deleted element = %d", e);
    printf("\n new array is\n");
     for ( i=0; i<n; i++)
      printf("%d", a[i]);
  }
void main()
  {
     int  a[30], i, k, n;
     printf("Enter no. of elements");
      scanf("%d", &n);
     printf("Enter array elements");
      for ( i=0; i<n; i++)
       scanf("%d", &a[i])
     printf("Enter location of item to be de
     scanf("%d", &k);
      delete ( a, n, k);
  }
```

2D array or matrix-

Declaration-

datatype arrayname [row][column]

int a[2][3];

Input →

```
for(i=0; i<row; i++)
    for(j=0; j<column; j++)
        scanf("%d", &a[i][j]);
```

Output →

```
for(i=0; i<row; i++)
    for(j=0; j<row; j++)
        printf("%d", a[i][j];
```

Representation of 2D array in memory-

- 2D array is stored in memory in two ways:
- row major- If the array elements are stored as row by row, it is called row major order.
- column major- If the array elements are stored as column by column, it is called column major order.

Example→ consider the array

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

row major order = {a, b, c, d, e, f, g, h, i}

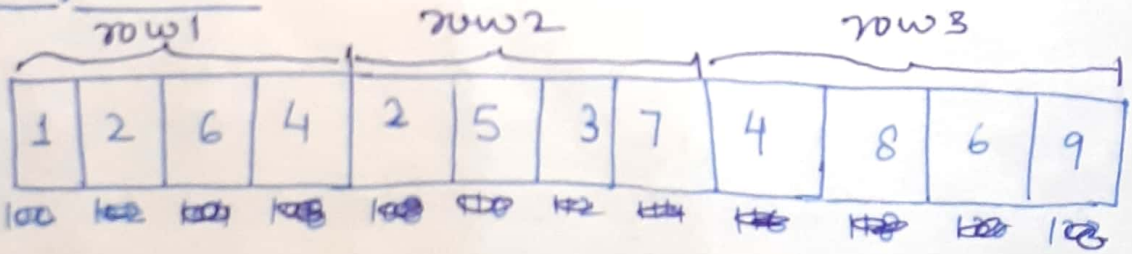
column major order = {a, d, g, b, e, h, c, f, i}

Q:→ show how the matrix $\begin{bmatrix} 1 & 2 & 6 & 4 \\ 2 & 5 & 3 & 7 \\ 4 & 8 & 6 & 9 \end{bmatrix}$ would appear in memory when stored in

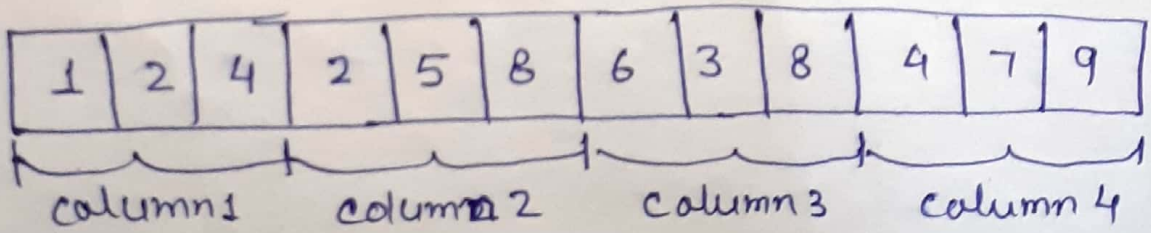
a) row major

b) column major

Sol:- row major order



column major order



Q:- A 2D array with 3 rows and 4 columns stored in memory as :

2 1 3 4 7 8 5 6 3 2 1 5

show the matrix form as it is stored in row major and column major.

Sol:- row major

$$\begin{bmatrix} 2 & 1 & 3 & 4 \\ 7 & 8 & 5 & 6 \\ 3 & 2 & 1 & 5 \end{bmatrix}$$

Column major-

$$\begin{bmatrix} 2 & 4 & 5 & 2 \\ 1 & 7 & 6 & 1 \\ 3 & 8 & 3 & 5 \end{bmatrix}$$

Address calculation for 2D array-

length = UB - LB + 1

$L_i = UB_i - LB_i + 1$

Effective index $E_i = i - LB$

row major →

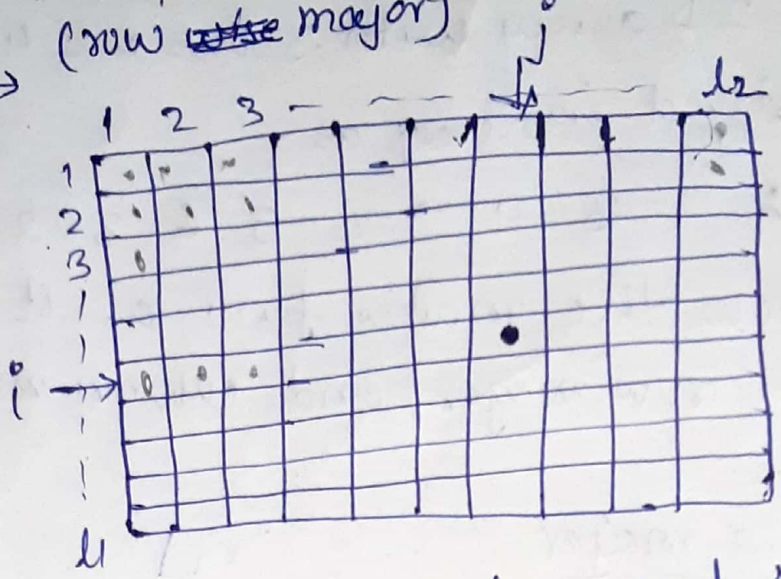
$loc(A[i][j]) = base(A) + w(E_1 L_2 + E_2)$

column major

$loc(A[i][j]) = base(A) + w(E_2 L_1 + E_1)$

Pradeep Pant

(4) Derivation → (row ~~wise~~ major)



let base address of array is α and size is 1 byte.
l_1 and l_2 is the length of row and column index

$$A[1][1] = \alpha$$

$A[1][2] = \alpha + 1$

$A[1][3] = \alpha + 2$

$A[1][l_2] = \alpha + (l_2 - 1)$

$$A[2][1] = \alpha + l_2$$

$A[2][2] = \alpha + l_2 + 1$

$A[2][3] = \alpha + l_2 + 2$

$A[2][l_2] = \alpha + l_2 + (l_2 - 1)$

$= \alpha + 2l_2 - 1$

$$A[3][1] = \alpha + 2l_2$$

$A[i][1] = \alpha + (i-1)\, l_2$

$A[i][2] = \alpha + (i-1)\, l_2 + 1$

$A[i][3] = \alpha + (i-1)\, l_2 + 2$

$A[i][j] = \alpha + (i-1)\, l_2 + (j-1)$

$A[i][j] = \alpha + 1.[E_1 l_2 + E_2]$

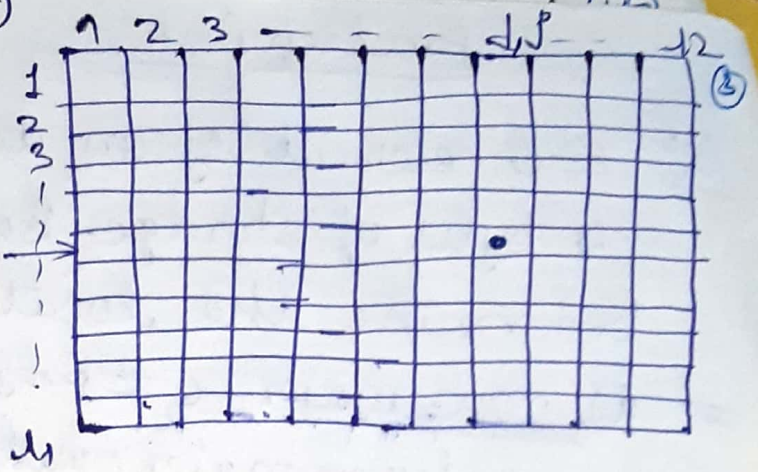
now if size = w

$$A[i][j] = \alpha + w(E_1 l_2 + E_2)$$

Proved

column major

(5)



let baseaddress = α
and size = 1 Byte

$$\boxed{A[1][1] = \alpha}$$

$A[2][1] = \alpha + 1$

$A[3][1] = \alpha + 2$

⋮

$A[l_1][1] = \alpha + (l_1 - 1)$

$$\boxed{A[1][2] = \alpha + l_1}$$

$A[2][2] = \alpha + l_1 + 1$

$A[3][2] = \alpha + l_1 + 2$

⋮

$A[l_1][2] = \alpha + l_1 + (l_1 - 1)$

$A[l_1][2] = \alpha + 2l_1 - 1$

$$\boxed{A[1][3] = \alpha + 2l_1}$$

~~A[1][3]~~ !

$A[1][j] = \alpha + (j-1)\,l_1$

$A[2][j] = \alpha + (j-1)\,l_1 + 1$

$A[3][j] = \alpha + (j-1)\,l_1 + 2$

⋮

$A[i][j] = \alpha + (j-1)\,l_1 + (i-1)$

$$\boxed{A[i][j] = \alpha + E_2 l_1 + E_1}$$

if size w

$$\boxed{Loc(A[i][j]) = \alpha + w(E_2 l_1 + E_1)}$$

Proved

Pradeep Pant

Pradeep Pant (b)

Q:→ Each element of an array A[20][50] requires 4 bytes of storage. Base address is 2000. Determine the location A[10][10] in
a) row major (c programming)
b) column major (Matlab, python Pandas, numpy)

Sol:→ $d_1 = UB_1 - LB_1 + 1 = 20 - 1 + 1 = 20$

$d_2 = UB_2 - LB_2 + 1 = 50 - 1 + 1 = 50$

$E_1 = i - 1 = 10 - 1 = 9$

$E_2 = j - 1 = 10 - 1 = 9$

a) row major-

$$Loc(A[10][10]) = Base(A) + W(E_1 d_2 + E_2)$$
$$= 2000 + 4 * (9 \times 50 + 9)$$
$$= 2000 + 4(459)$$
$$= 3836$$

b) column major-

$$Loc(A[10][10]) = Base(A) + W(E_2 d_1 + E_1)$$
$$= 2000 + 4 \times (9 \times 20 + 9)$$
$$= 2000 + 4 \times (189)$$
$$= 2756$$

Pradeep Pant

Q:- Given an array A[6][6] whose first element is stored at location 100. Calculate the location of A[2][5] if each element is stored in 4 bytes and array is stored row wise.

Ans:-
$$l_1 = UB_1 - LB_1 + 1 = 6 - 1 + 1 = 6$$
$$l_2 = UB_2 - LB_2 + 1 = 6 - 1 + 1 = 6$$
$$E_1 = i - LB_1 = 2 - 1 = 1$$
$$E_2 = j - LB_2 = 5 - 1 = 4$$

On row major

$$Loc(A[2][5]) = base(A) + W * (E_1 l_2 + E_2)$$
$$= 100 + 4 \times (1 \times 6 + 4)$$
$$= 140$$

- Calculate the address of X[4,3] in a 2D array X(1...5, 1...4) stored in a row major order. Assume the base address to be 1000 and each element require 4 words of storages.

Ans:-
$$l_1 = UB_1 - LB_1 + 1 = 5 - 1 + 1 = 5$$
$$l_2 = UB_2 - LB_2 + 1 = 4 - 1 + 1 = 4$$
$$E_1 = i - LB_1 = 4 - 1 = 3$$
$$E_2 = j - LB_2 = 3 - 1 = 2$$
$$Loc(X[4,3]) = 1000 + 4 * (3 \times 4 + 2)$$
$$= 1000 + 56 = 1056$$

Pradeep Pant

Q:- An array X[-15....10, 15....40] requires (8) 1 Byte of storage. if begining location is 1500. Determine the location of X[15][20].

Soln

$l_1 = UB_1 - LB_1 + 1 = 10 - (-15) + 1 = 26$

$l_2 = UB_2 - LB_2 + 1 = 40 - 15 + 1 = 26$

$E_1 = i - LB_1 = 15 - (-15) = 30$

$E_2 = j - LB_1 = 20 - 15 = 5$ X[15][20] is out of bounds

row major as upper bound of first ~~not~~ dimension is 10.

$Loc[X[15][20]) = 1500 + (30 \times 26 + 5)$

$= 1500 + 785 = 2285$

column major

$loc(X[15][20]) = 1500 + (5 \times 26 + 30)$

$= 1500 + 160 = 1660$

multidimensional Array -

i) row major

$Loc(A[i][j][k]...[n]) = Base(A) + w[E_1 l_2 l_3 .. l_n + E_2 l_3 l_4 .. l_n + E_3 l_4 l_5 .. l_n + ... + E_n]$

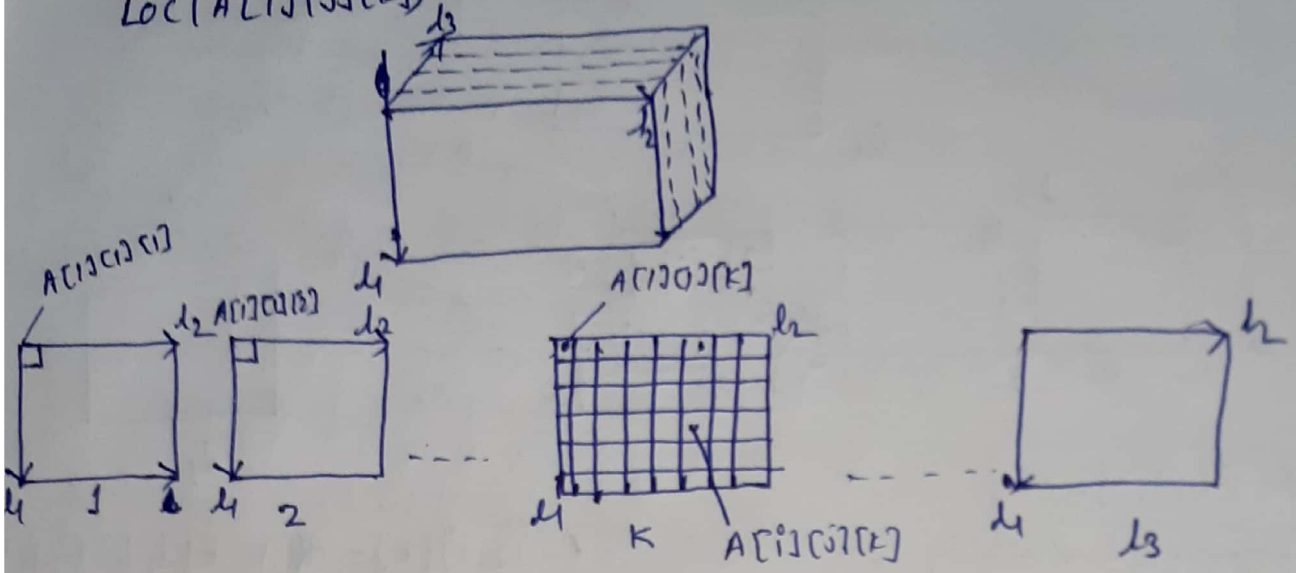
ii) column major -

$Loc(A[i][j][k]..[n]) = Base(A) + w[E_n l_{n-1} l_{n-2} .. l_1 + E_{n-1} l_{n-2} .. l_1 + ... + E_3 l_2 l_1 + E_2 l_1 + E_1]$

Derivation (3D array Address calculation)

column major

$$LOC(A[i][j][k]) = base(A) + w * (E_3 l_1 l_2 + E_2 l_1 + E_1)$$



~~A[1][1][1] = α~~

~~A[1][1][2] = α+1~~

~~A[1][1][3] = α+2~~

!

~~A[i][j][k] =~~

$$A[1][1][1] = \alpha$$

$$A[1][1][2] = \alpha + l_1 l_2$$

$$A[1][1][3] = \alpha + 2 l_1 l_2$$

!

$$A[1][1][k] = \alpha + (k-1) l_1 l_2$$

$$A[2][1][k] = \alpha + (k-1) l_1 l_2 + 1$$

$$A[3][1][k] = \alpha + (k-1) l_1 l_2 + 2$$

!

$$A[l_1][1][k] = \alpha + (k-1) l_1 l_2 + (l_1 - 1)$$

$$A[1][2][k] = \alpha + (k-1) l_1 l_2 + l_1$$

$$A[2][2][k] = \alpha + (k-1) l_1 l_2 + l_1 + 1$$

$$A[3][2][k] = \alpha + (k-1) l_1 l_2 + l_1 + 2$$

!

$$A[l_1][2][k] = \alpha + (k-1) l_1 l_2 + l_1 + l_1 - 1$$

$$A[l_1][2][k] = \alpha + (k-1) l_1 l_2 + 2 l_1 - 1$$

$$A[1][3][k] = \alpha + (k-1) l_1 l_2 + 2 l_1$$

!

$$A[1][j][k] = \alpha + (k-1) l_1 l_2 + (j-1) l_1$$

$$A[2][j][k] = \alpha + (k-1) l_1 l_2 + (j-1) l_1 + 1$$

$$A[3][j][k] = \alpha + (k-1) l_1 l_2 + (j-1) l_1 + 2$$

!

$$A[i][j][k] = \alpha + (k-1) l_1 l_2 + (j-1) l_1 + (i-1)$$

$$A[i][j][k] = \alpha + w(E_3 l_1 l_2 + E_2 l_1 + E_1)$$

~~similarly~~

~~A[i][j] = α + w(E2 l1 + E1)~~

$$A[i][j] = \alpha + w(E_2 l_1 + E_1)$$

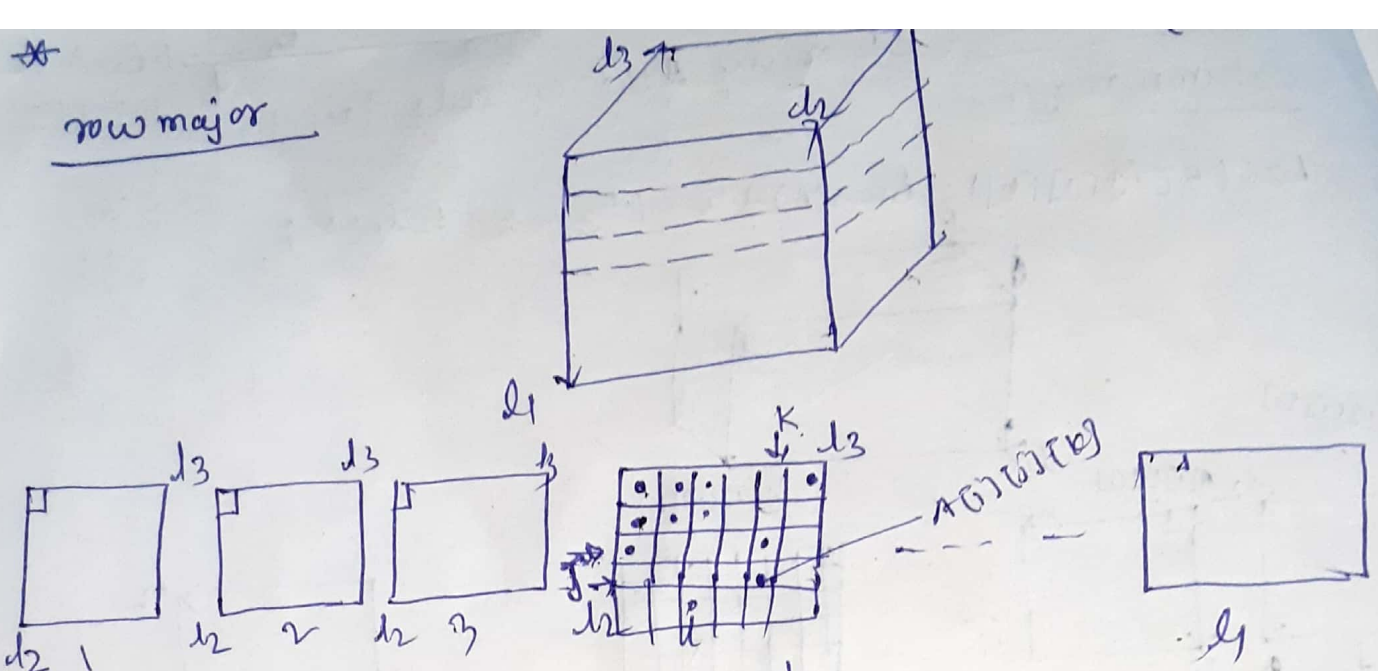
$$A[i][j][k] = \alpha + w(E_3 l_2 l_1 + E_2 l_1 + E_1)$$

!

similarly

$$A[i][j][k]\ldots[n] = \alpha + w[E_n l_{n-1} l_{n-2} \ldots l_1 + E_{n-1} l_{n-2} l_{n-3} \ldots l_1 + \ldots + E_2 l_1 + E_1]$$

row major



$A[1][1][1] = \alpha$

$A[2][1][1] = \alpha + l_2 l_3$

$A[3][1][1] = \alpha + 2 l_2 l_3$

$A[i][1][1] = \alpha + (i-1) l_2 l_3$

$A[i][1][2] = \alpha + (i-1) l_2 l_3 + 1$

$A[i][1][3] = \alpha + (i-1) l_2 l_3 + 2$

$A[i][1][l_3] = \alpha + (i-1) l_2 l_3 + (l_3 - 1)$

$A[i][2][1] = \alpha + (i-1) l_2 l_3 + l_3$

$A[i][2][2] = \alpha + (i-1) l_2 l_3 + l_3 + 1$

$A[i][2][3] = \alpha + (i-1) l_2 l_3 + l_3 + 2$

$A[i][2][l_3] = \alpha + (i-1) l_2 l_3 + l_3 + l_3 - 1$

$= \alpha + (i-1) l_2 l_3 + 2 l_3 - 1$

$A[i][3][1] = \alpha + (i-1) l_2 l_3 + 2 l_3$

$A[i][j][1] = \alpha + (i-1) l_2 l_3 + (j-1) l_3$

$A[i][j][2] = \alpha + (i-1) l_2 l_3 + (j-1) l_3 + 1$

$A[i][j][3] = \alpha + (i-1) l_2 l_3 + (j-1) l_3 + 2$

$A[i][j][k] = \alpha + (i-1) l_2 l_3 + (j-1) l_3 + (k-1)$

$A[i][j][k] = \alpha + E_1 l_2 l_3 + E_2 l_3 + E_3$

if size = w

$A[i][j][k] = \alpha + w(E_1 l_2 l_3 + E_2 l_3 + E_3)$

similarly

$A[i][j][k]\ldots[n] = \alpha + w[E_1 l_2 l_3 \ldots l_n + E_2 l_3 l_4 \ldots l_n + \ldots + E_{n-1} l_n + E_n]$

Q1 Consider a 3D array A[2:8, -4:1, 6:10] where base address is 200. and w is 4 words per memory cell. calculate the address of A[5, -1, 8] for row major order and column major order. Also find the length of array.

Sol1 $l_1 = UB_1 - LB_1 + 1 = 8 - 2 + 1 = 7$

$l_2 = UB_2 - LB_2 + 1 = 1 - (-4) + 1 = 6$

$l_3 = UB_3 - LB_3 + 1 = 10 - 6 + 1 = 5$

Length of array $= l_1 . l_2 . l_3 = 7 \times 5 \times 5 = 210$

$E_1 = i - LB_i = 5 - 2 = 3$

$E_2 = j - LB_j = -1 - (-4) = 3$

$E_3 = k - LB_k = 8 - 6 = 2$

row major

$Loc(A[5, -1, 8]) = base(A) + w * (E_1 l_2 l_3 + E_2 l_3 + E_3)$

$= 200 + 4 * (3 \times 6 \times 5 + 3 \times 5 + 2)$

$= 200 + 4 (90 + 15 + 2)$

$= 200 + 4 \times 107 = 200 + 428 = 628$

~~Let takes a multidimensional array A[8][7][10][6]~~

column major

$Loc(A[5, -1, 8]) = base(A) + w (E_3 l_2 l_1 + E_2 l_1 + E_1)$

$= 200 + 4 * (2 \times 6 \times 7 + 3 \times 7 + 3)$

$= 200 + 4 * (84 + 21 + 3)$

$= 200 + 4 * (108)$

$= 200 + 432$

$= 632$

Pradeep Pant

Q:- Let's take a multidimensional array A[8][10][10][20] with the base address 1000, calculate address of A[2][3][5][6]. by row major order. take w=2

Sol:- ~~Loc(A)~~ $l_1 = 2-1 = 1$ $l_2 = 10-1 = 9$ $l_3 = 10-1 = 9$ $l_4 = 20-1 = 19$

$E_1 = 2-1 = 1$ $E_2 = 3-1 = 2$ $E_3 = 5-1 = 4$ $E_4 = 6-1 = 5$

$Loc(A[2][3][5][6]) = 1000 + 2*(5\times 1\times 9\times 9 + 4\times 1\times 9 + 2\times 1 + 1)$

$= 1000 + 2*(405 + 36 + 2 + 1)$

$= 1000 + 2*(444)$

$= 1000 + 888 = \underline{1888}$

Sparse matrix- if number of zero elements are greater than number of non zero elements in a 2D array or matrix. This type of matrix is known as sparse matrix.

example-

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 \\ 5 & 0 & 2 & 5 \end{bmatrix}$$

Representation of sparse matrix

Sparse matrix representation can be done by two ways

1) Array representation
2) Linked List representation.

1) Using Arrays- 2D array is used to represent a sparse matrix in which there are three rows named as

- Row – index of row, where non zero element is located.
- column – index of column, where non zero element is located.
- value – value of non zero element located at index. (row, column).

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 2 & 0 & 0 & 0 \\ 5 & 0 & 2 & 5 \end{bmatrix} \end{array} \Rightarrow$$

row	0	1	2	3	3
column	4	1	1	3	4
value	1	2	5	2	5

2) Using Linked list –

program →

```
void main()
{
  int A[3][2] = {{1,0},{0,0},{0,1}};
  int n=0; i, j;
  for(i=0; i<3; i++)
    for(j=0; j<2; j++)
    {
      if(A[i][j] != 0)
        n++;
    int S[3][n]; k=0;
  for(i=0; i<3; i++)
  {
    for(j=0; j<2; j++)
    {
```

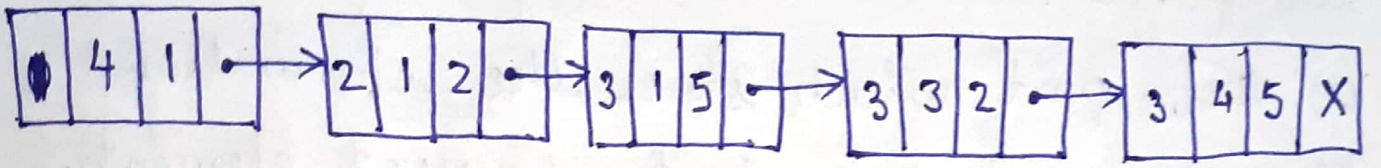
```
if (A[i][j] != 0)
 {
    S[0][k]= i;
    S[1][k]=j;
    S[2][k] = A[i][j];
     k++;
  }
 }
}
for (i=0; i<3, i++)
 {
   for (j=0;j<n;j++)
     printf("%d ", S[i][j]);
   printf("\n");
 }
}
```

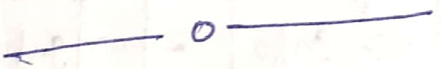
2) Using Linked List:- In linked list each node has four fields. These are

- Row - Index of row, where non zero element is located.
- column - Index of column, where non zero element is located.
- value - Value of non zero element located at index (row, column)
- next node - Address of next node.

	1	2	3	4
1	0	0	0	1
2	2	0	0	0
3	5	0	2	5

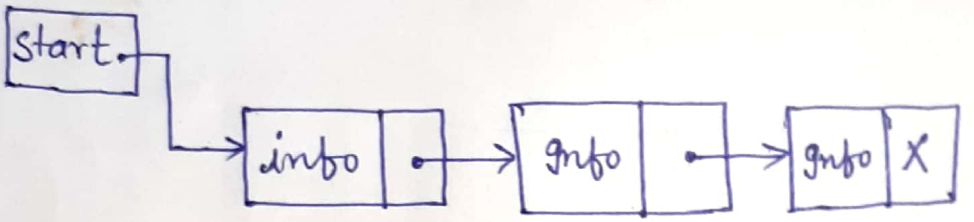
$\Rightarrow$ ..





Linked List

- Linked list are special list of some data element linked to one another.
- Each element is called a node which has two parts
  a) Info part which stores the information.
  b) pointer which points to the next elements.



These nodes are structure containing two fields
(i) Data field ii) Link field

The link field of last node contains NULL.

Linked list is linear or non linear

→ According to accessing the element linked list is linear.
→ According to storage linked list is non linear.

Application of Linked List:-

- Arithmatic operations can be easily performed.
- manipulation and evaluation of polynomial is easy.
- Useful in symbol table construction.

Advantages-

1) Linked list are dynamic data structure, It can grow or shrink during program execution.
2) Efficient memory utilization - Here memory is not pre allocated. Memory is allocated whenever it is required and it is deallocated when no longer needed.

3) Insertion and Deletion are easier and efficient.

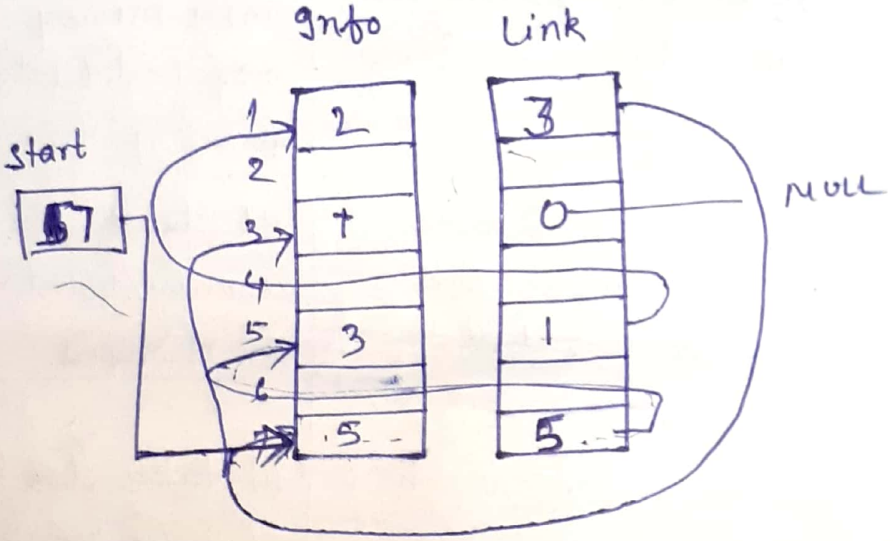
Disadvantages-

1) more memory- no. of fields are more, so more memory required.

2) Access to an element is time consuming.

Representation of Linked List in memory-

List requires two linear array Info and Link. List also requires a variable start which contains the location of the begining of the list. and nextpointer sentinel, denoted by NULL, which indicates the end of list.



Types of Lined List-

1) Singly Linked List or One way list
2) Circular Singly Linked List
3) Doubly Linked List
4) Circular doubly Linked List
5) Header Linked List

Pradeep Pant

~~singly linked list~~

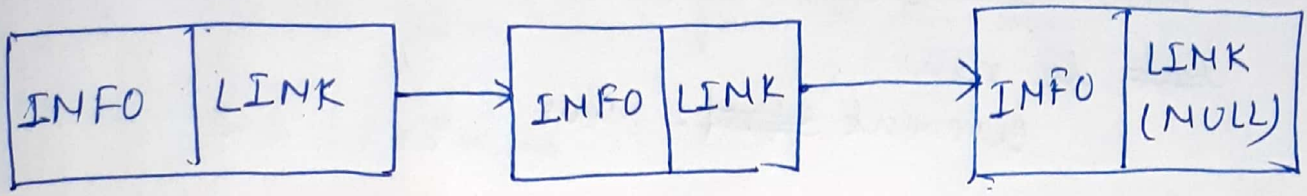
Difference between Array and Linked List

Characteristics	Array	Linked list
memory Allocation	static	Dynamic
memory Deallocation	automatically	free() function is used for deallocation
element access	randomly so less time taken	sequentially so more time taken
storage (memory requirement)	if no. of elements are same then less memory required	more memory required. (data and pointer)
Insertion	it takes more time as all element right to the desired location are shifted towards right.	it takes less time as only pointer part is changed.
Deletion	it takes more time as all elements right to the desired location are shifted towards left.	it takes less time as only pointer part is changed

Singly linked list (one way list or linear)

In singly linked list each node is divided into two parts

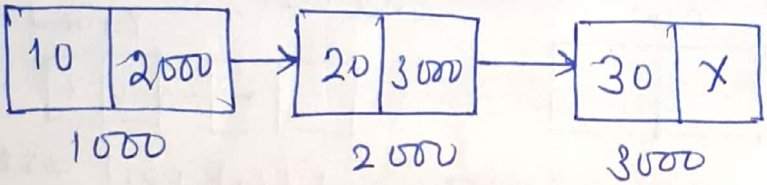
- information or data (INFO)
- address of next node (LINK)



first node

last node

ex→

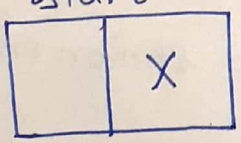


Operations on Singly linked list-

create() →

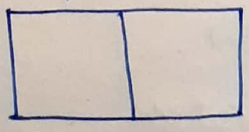
Concept → create a node and called this node as start node and initialize is LINK part as NULL.

start



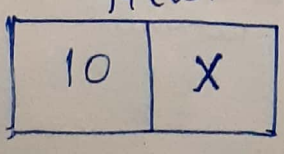
Create a new node and called it New.

New



Initialize its info part as 10 and Link part as NULL.

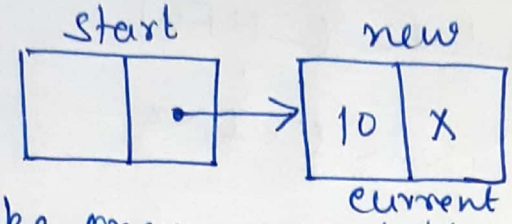
New.



Pradeep Pant

Now check the LINK part of start node. (20)
if (LINK[start] = NULL) then make

LINK[start] = new



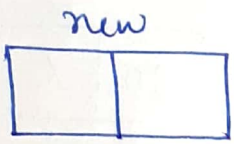
——————— (1)

Also take one more variable current and set it ~~not~~ to new

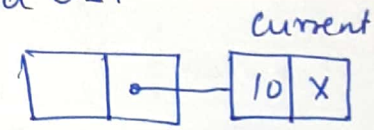
current ~~start~~ = New

for M = 2

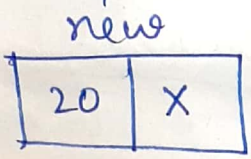
create a new node



and List



Initialize its INFO part as 20 and Link part as NULL



Now check the Link part of start node..
If Link[start] = NULL then make

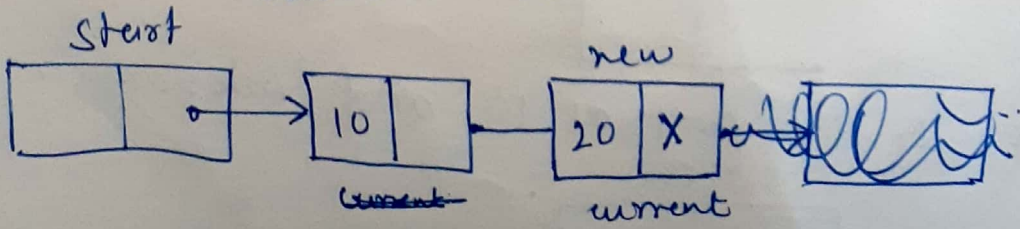
Link[start] = new

~~otherwise~~ In the above shown in (1) LINK[start] = NEW

else

current → next = new

current = new



repeat this process for desired N.

Algorithm →

createSinglyLinkedList(N)

1. set link[start] = NULL, i = 1
2. repeat step 3 to 8 while (i <= N)
3. create new node from AVAIL List.
4. set INFO[new] = item
5. set LINK[new] = NULL.
6. if (LINK[start] = NULL) then
   LINK[start] = new
   current = new
7. else
   LINK[current] = new
   current = new
8. Set i = i + 1
   [End of step 2 loop]
9. Exit.

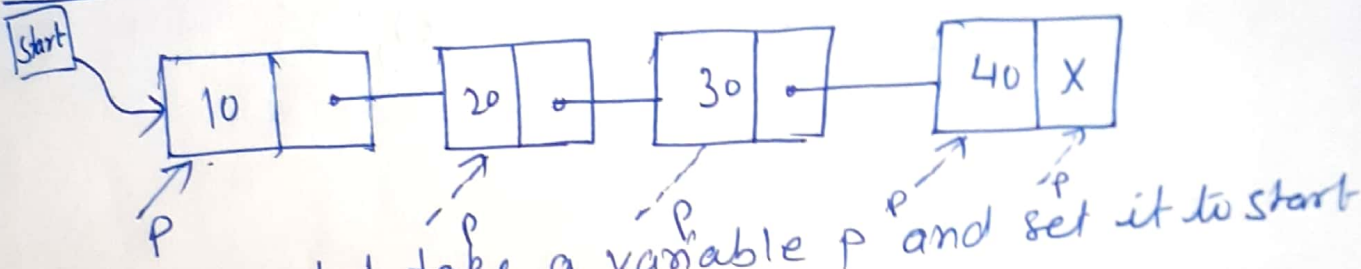
C function →

```
struct node
{
    int info;
    struct node *next;
}*start = NULL;

void create()
{
    struct node *new, *start;
    int N, i, item;
    printf("Enter No. of nodes");
    scanf("%d", &N);
    for (i=1; i<=N; i++)
    {
        new = (struct node *)malloc(sizeof(struct node
        printf("Enter item");
        scanf("%d", &item);
        new->info = item;
        scanf("%d", &new->data)
        new->next = NULL;
        if (start == NULL)
        {
            start = new
            current = new
        }
```

```
else
{
  current -> next = new
      current = new
}

}
```

2) Traversing →

concept →



Let take a variable p and set it to start

p = start

process INFO[P]

set P = ~~p->next~~ LINK[P]

repeat above two steps until P = NULL

Algorithm → Traverse()

~~Let list be an linked~~

1. Set p = start
2. if (p = NULL)
   write "List is Empty"
   goto step 6
   [End of if]
3. Repeat step 4 and 5 until p != NULL
4. process INFO[p]
5. set P = ~~P->~~ LINK[P]

Prabudh Pant

[End of step 5 loop]

6. Exit

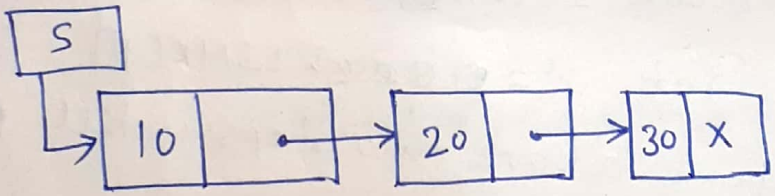
C function →

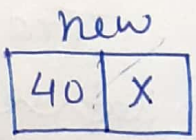
```
void display()
{
  struct node *p;
printf("\n The list is : ");
for( p = start ; p != NULL ; p = p->next)
    printf(" %d", p->info);
}
```

3) Insertion -

a) Insertion at begining -

concept →

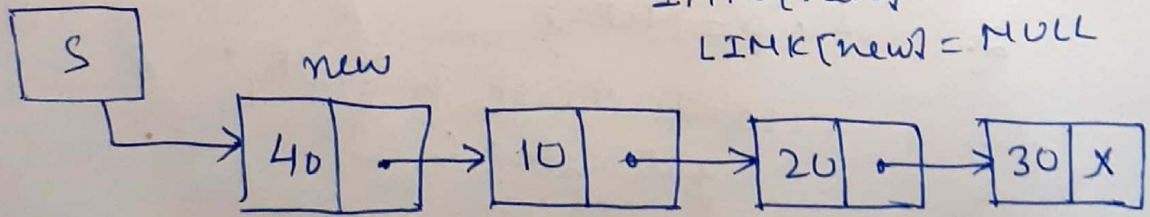




create new node from AVAIL LIST

INFO[new] = item

LINK[new] = NULL



LINK[new] = start.

start = new

Algorithm →

Insertatbegining( )

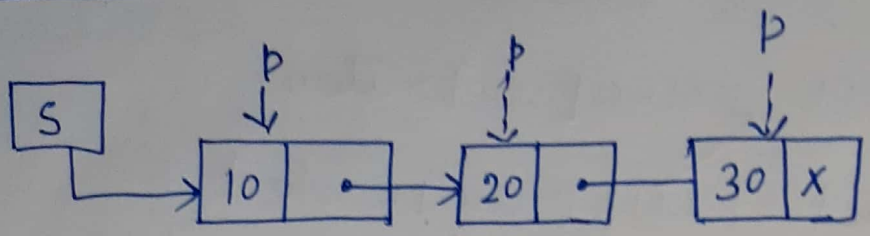
1. Set new = AVAIL AVAIL = LINK[AVAIL]
2. Set INFO[new] = item
3. Set LINK[new] = start
4. start = new
5. Exit

function -

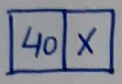
```
void insertatbegining( )
{
   struct node *new;
   printf("Enter node to insert");
   new = (struct node *) malloc (sizeof (struct node));
   new->info=
    printf("Enter data");
    scanf("%d", &new->info);
     new->next = start;
      start = new;
printf("Item inserted succesfully");
}
```

b) Insertion at last:→

concept→



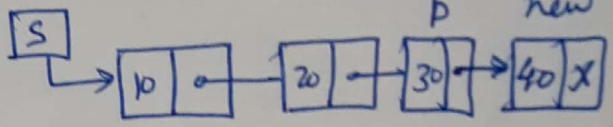
new



set p = start traverse list untill LINK[P] != NULL

now set p -> next = new

new -> next = NULL



Algorithm→

insert_at_last( )

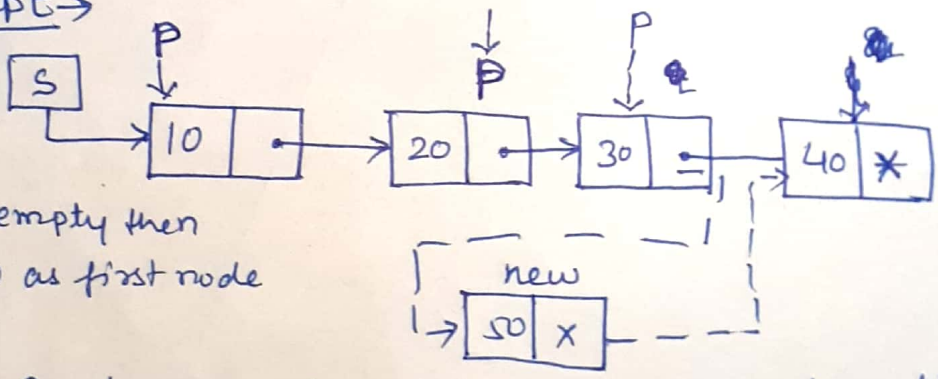
1. set new = AVAIL and AVAIL = LINK[AVAIL]
2. set INFO[new] = item
3. set p = start
4. Repeat step 5 until LINK[P] != NULL
5. P = LINK[P]
6. LINK[P] = new
7. LINK[new] = NULL
8. exit

a function

```
void insert_at_last()
{
    struct node *new, *p;
    printf("Enter node to insert");
    new=(struct node *) malloc(sizeof(struct node));
    printf("Enter data");
    scanf("%d", &new->info);
    for(p=start; p->next!=NULL; p=p->next);
      p->next=new;
      new->next=NULL
    printf("data inserted successfully");
}
```

c) Insert after node :-

concept →



If list is empty then insert new as first node

t = traverse the list unhill the node after which new node is inserted. input data of node

i.e. for(p=start; p->info != data; p=p->next)

now we find p ~~for q=p->next~~

now ~~p->next=new~~ new->next = p->next

~~new->next=q~~ p->next = new

Pradip Pant

Algorithm

Insert_after_node( )

1. new = AVAIL and AVAIL = LINK[AVAIL]
2. INFO[NEW] = item
3. read ~~data~~ into of a node after which node is inserted in variable data
4. if (start = NULL) then
   start = new
   LINK[new] = NULL
   [END of if]
5. else
   ~~for(p=start~~
   Set p = start
   Repeat while (INFO[p] != data)
   p = p -> next
   [End of else
6. new -> next = p -> next
7. p -> next = new
8. Exit

c function

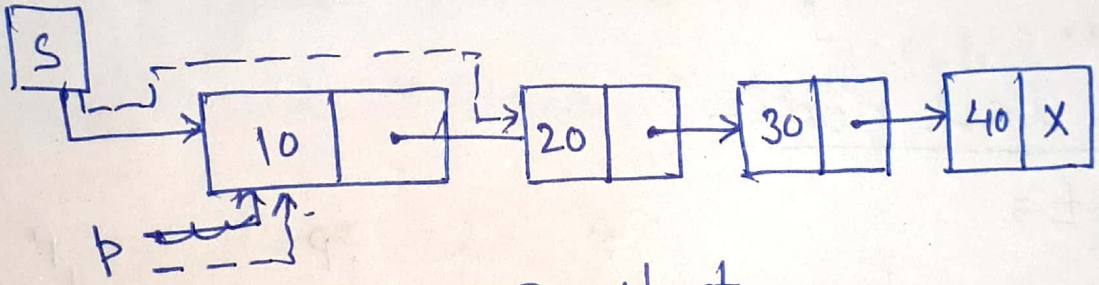
```
void insert_after_node()
{
    struct node *new, *p;
    int data;
    printf("Enter node to insert");
new=(struct node *) malloc(sizeof(struct node));
    printf("Enter info of new node");
    scanf("%d", &new->info);
    if(start==NULL)
    {
        start=new;
        new->next=NULL;
    }
    else
    {
        printf("Enter info of data after inserted a new node");
        scanf("%d", &data);
        for(p=start; p->info!=data; p=p->next);
        new->next=p->next;
        p->next=new;
        printf("node inserted sucessfully");
    }
```

Scanned with CamScanner

4) Deletion → Deletion means to delete a node from the list. Deletion is done by three ways.

a) Delete first Node
b) Delete Last Node
c) Delete a perticular Node. (Except first and last)

a) Delete first node :- concept



Let P = start
start = P → next
Deallocate P

Algorithm →

Delete_first ( )

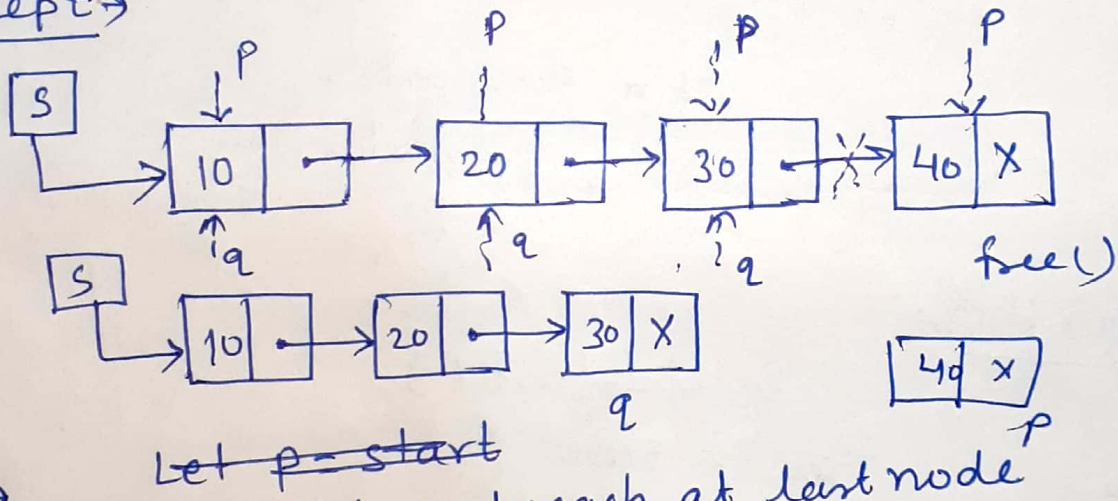
1. set P = start
2. data = INFO[P]
3. start = P → next
4. Print data
5. free (P)
6. Exit

C function

```
void delete()
{ int data;
  struct node *p;
    p = start;
  data = p -> info;
  start = p -> next;
  free(p);
}
```

b) Delete last node-

concept→



~~Let p = start~~

traverse list and reach at last node

for(p = start; p -> next != NULL; p = p -> next)

q = p

q p and q both move simultaneously but q stops just before the last node because at p -> next = NULL (last node) loop condition is false so q = p is not executed.

data = ~~P~~ INFO(P)

now free(P)

and q -> next = NULL

Algorithm

delete_last()

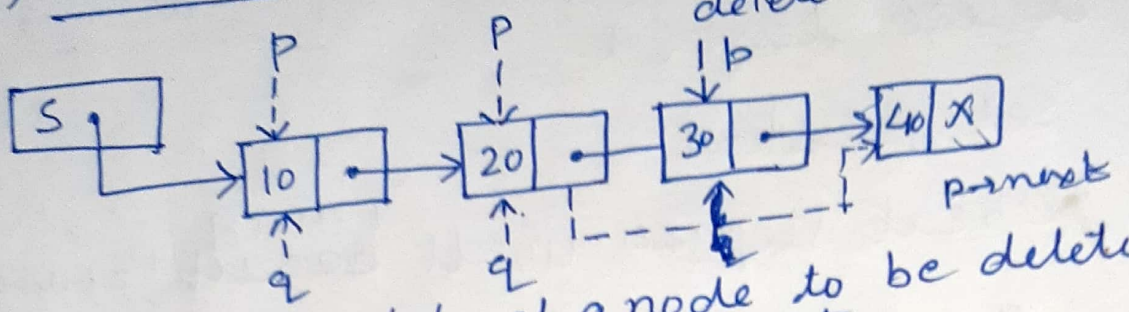
1. set P = start
2. Repeat step3 & 4 untill LINK[P] != NULL
3. set q = P
4. set P = LINK[P]
5. set data = INFO[P]
6. write a data, " is deleted "
7. free(P)
8. LINK[q] = NULL
9. Exit

unction →

```
void delete_last()
{
struct node *p, *q;
  int d;
for(p=start; p->next!=NULL; p=p->next)
    q=p;
  d= p->info;
printf("%d is deleted", d);
   free(p);
 q->next = NULL;
```

}

c) delete after node -



input the info of a node to be deleted (data)
input data

```
for( P=start; P->info != data; P=P->next)
    q=P
```

now
```
q->next = P->next
free(P)
```

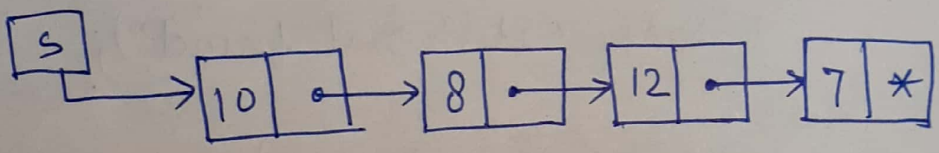
Algorithm → delete_after_node( )

1. Input data of a node to be deleted in variable data
2. set P = start
3. Repeat step 4 while (P->INFO[P] != data)
4. Set q = P
   [End of step 3 loop]
5. q->next = P->next
6. free(P)
7. write data, " is deleted"
8. Exit

c ~~program~~ function

```
void delete( )
{
 struct node *p, *q;
  int d;
 printf("Enter info of a node to be deleted");
 scanf("%d", d);
 for(p= start; p->info != d; p= p->next)
        q= p;
   q->next = p->next;
    free(p);
 printf("%d is deleted", d);
}
```

Searching → concept



let element to search = 12

traverse the list from first node to last

untill p->info != item and p != NULL

(item found) (item not found)

Algorithm →

search_list( )

1. input item to be searched
2. set p = start
3. Repeat step 4 untill (INFO[P] != item and P != NULL)
4. P = LINK[P]

[END of step 3 loop]

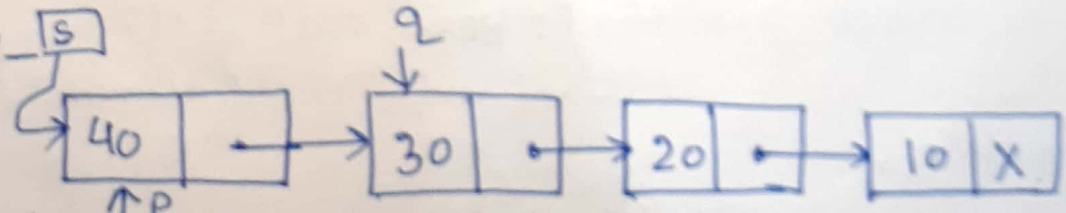
5. if (P==NULL)
   write "item not found"
   else
   write "item found"

6. Exit

c function-

```
void search_list()
{ struct node *p;
  int item;
  printf("Enter item to be searched");
  scanf("%d", &item);
  for (P= start; P->info != item && P != NULL;
                                P=P->next);
  if (P==NULL)
     printf("item not found");
  else
     printf("item found");
}
```

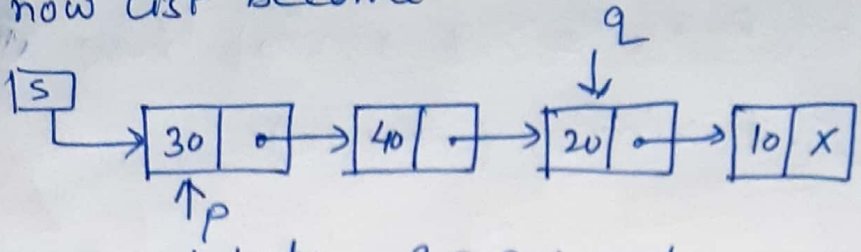
6) sorting-



Your task is sort the list in ascending order.

Set P= start
q = P->next
check if (P->info > q->info) then
swap(P->info, q->info)

now list become

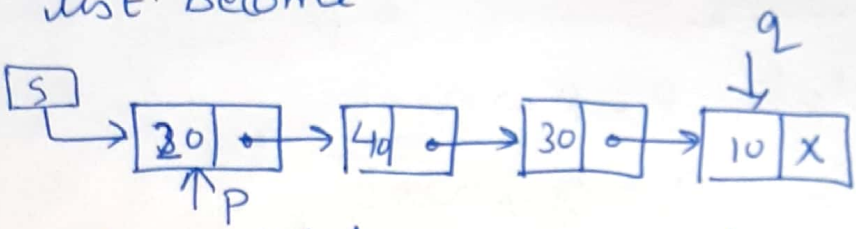


now update $q = q \rightarrow next$

again check ($p \rightarrow info > q \rightarrow info$) then

swap ($p \rightarrow info$, $q \rightarrow info$)

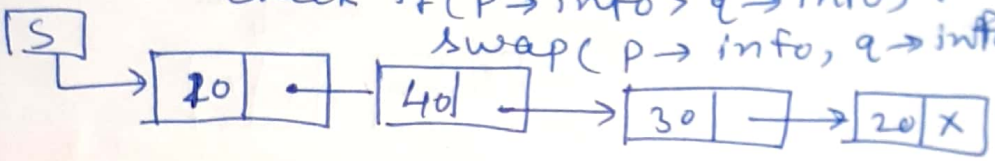
list become



now update $q = q \rightarrow next$

check if ($p \rightarrow info > q \rightarrow info$) then

swap ($p \rightarrow info$, $q \rightarrow info$)

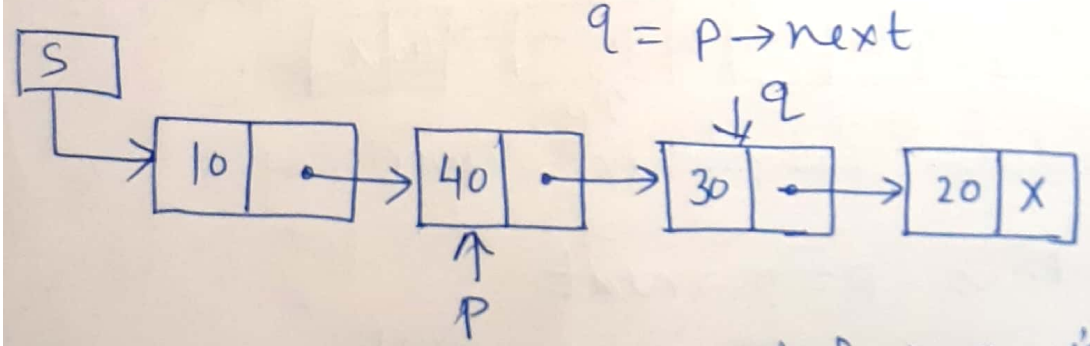


now update $q = q \rightarrow next = NULL$

no more node. what happened? The lowest element comes in first position.

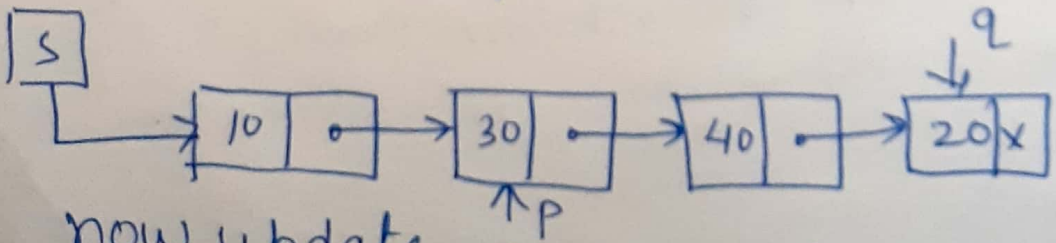
now update $p = p \rightarrow next$

$q = p \rightarrow next$



check if ($p \rightarrow info > q \rightarrow info$) then

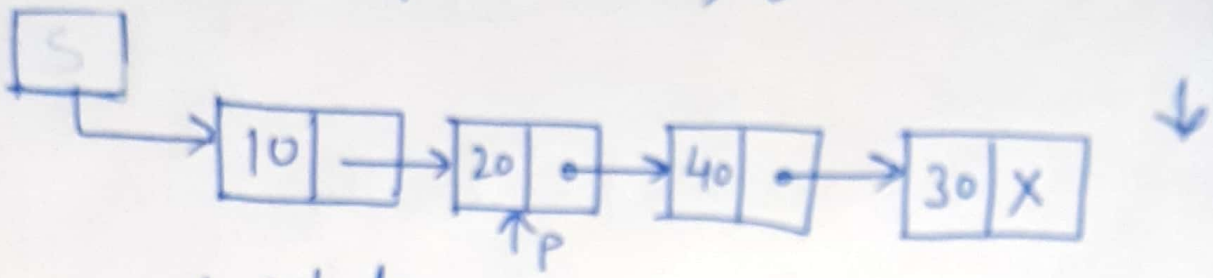
swap ($p \rightarrow info$, $q \rightarrow info$)



now update $q = q \rightarrow next$

check if (p→info > q→info) then

swap(p→info, q→info)

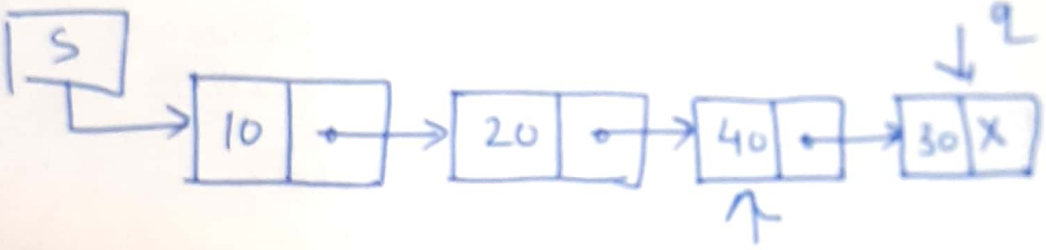


update q = q→next = NULL

no more nodes.

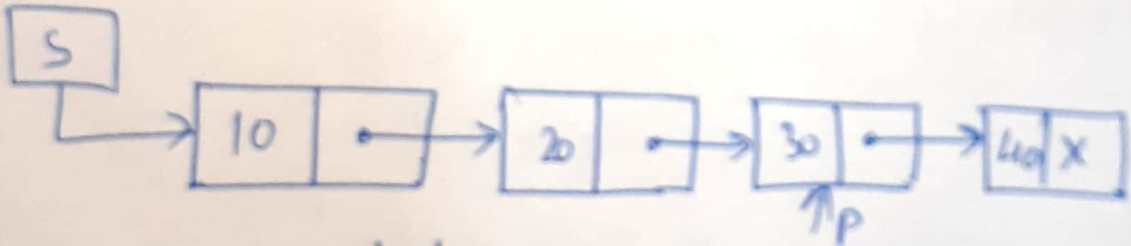
now update p = p→next

q = q→next



check if (p→info > q→info)

swap(p→info, q→info)



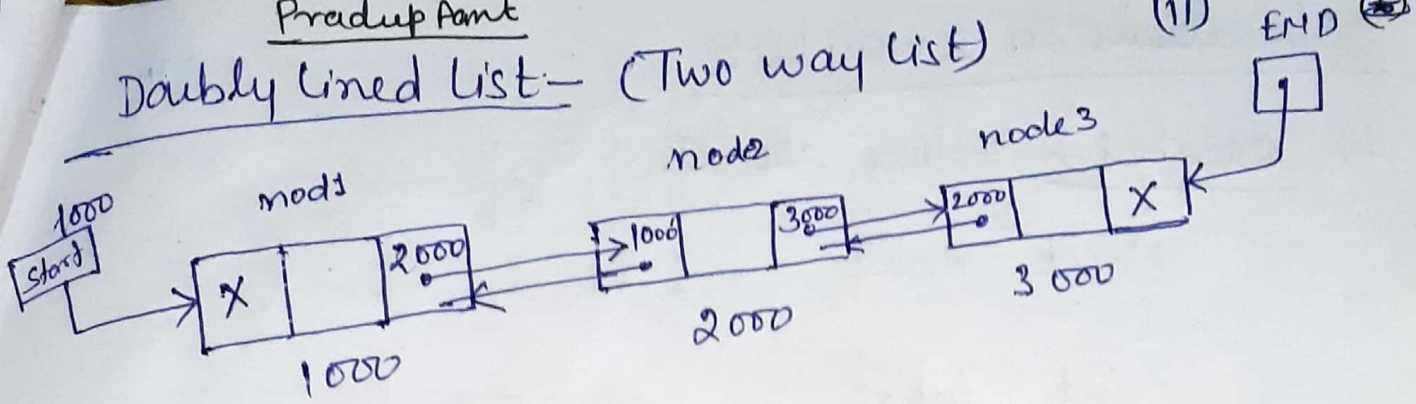
now update q = q→next = NULL

now update p = p→next

q = p→next = NULL

so no more comparision.

Doubly Linked List – (Two way List)



Define structure

```
struct node
{
   int info;
   struct node *prev;
   struct node *next;
} start = *NULL; *end = NULL;
```

A two way list is a linear collection of data elements called nodes. Each node has 3 parts.

1) An information field INFO
2) A pointer field PREV which contains the location of the preceding node.
3) A pointer field NEXT which contain the location of the next node.

The list also requires two list pointer variables: START and END

By using variable START and pointer NEXT we can traverse a two way list in the forward direction.

By using variable END and pointer PREV we can traverse a two way list in the backward direction.

Operations on 2 way list

1) create() Define structure

```
struct node
{
 int info;
 struct node *next;
 struct node *prev;
}*start=NULL;
```

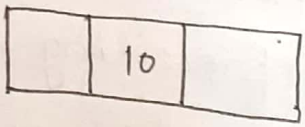
2) create()-

concept→ Take a new node

new = malloc()

Enter node info

new→info = item



~~check list is empty~~

~~if(start==NULL) then~~

Enter number of nodes in N.

check list is empty

if(start==NULL) then

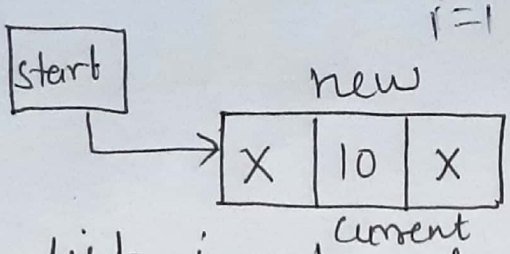
set start = new

start→next = NULL

start→prev = NULL

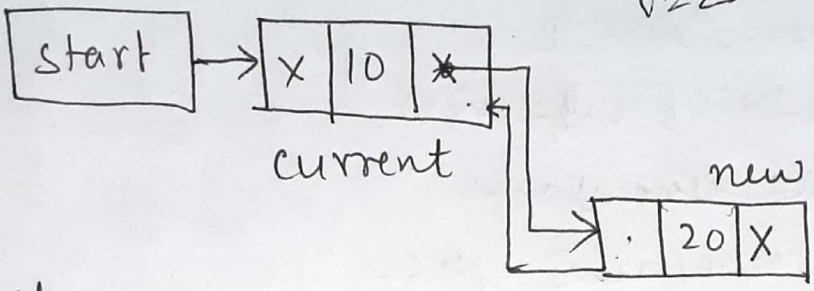
current = new

i=1

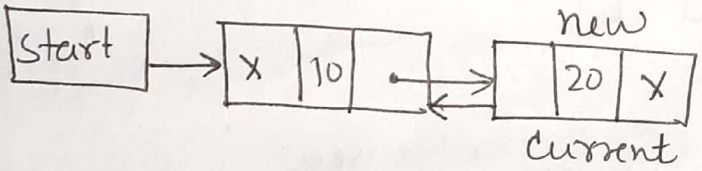


If list is not empty M=2

i=2

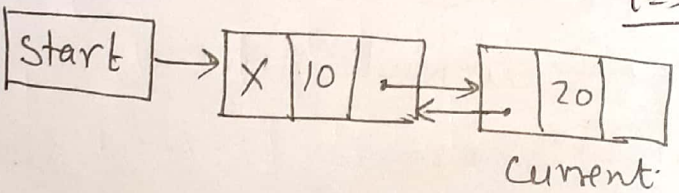


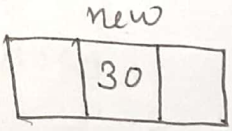

```
current -> next = new
new -> prev = current
new -> next = NULL
current = new
```



M = 3

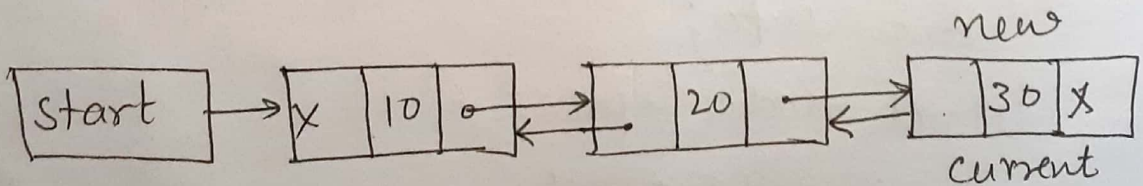
i=3





again

```
Current -> next = new
new -> prev = current
new -> next = NULL
current = new
```



Algorithm

Create( )

1. Input N
2. for (i=1 to N) repeat steps 3 to 7
3. new = AVAIL
4. INFO[new] = item
5. ~~LINK[new] = N~~
5. LINKP[new] = NULL
6. LINKN[new] = NULL
7. if (start = NULL) then
   start = new
   current = new
   [END of if]
   else
   LINKN[current] = new
   LINKP[new] = current
   ~~LINKN[]~~ current = new
   [END of else]
   [END of step 2 loop]
8. Exit

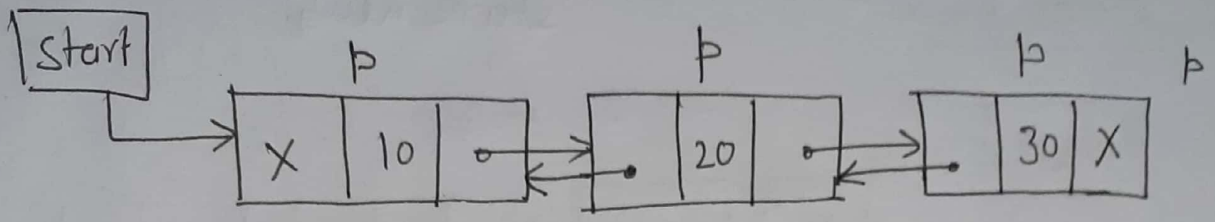
C function →

```
struct node
{
  int info;
struct node *prev;
struct node *next;
}*start = NULL;
```

```
void create()
{
  int i, N;
  struct node *new, *current;
  printf("Enter no. of nodes:");
  scanf("%d", &N);
  for(i=1; i<=N; i++)
  {
  new=(struct node *) malloc(sizeof(struct node));
   new->prev= NULL;
   new->next= NULL;
    if(start== NULL)
    {   start=new;
  current=new;
    }
  else
     {
       current->next=new;
       new->prev=current;
       current=new;
     }
   }

}
```

2) Traverse

concept→



set p = start

process P
set p = p → next } repeat these two statements until p ~~next != NULL~~ p != NULL

Algorithm→

traverse( )

1. set p = start
2. Repeat step 3 and 4 while (p != NULL)
3. Process P
4. p = LINKN[p]
   [END of step 2 loop]
5. Exit

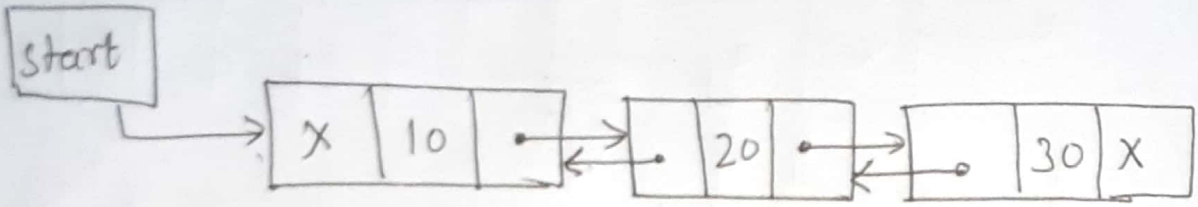
C function→

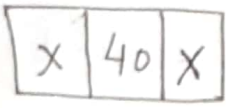
```
void traverse( )
{
  struct node *p;
  if (start == NULL)
    printf("empty list");
  for( p = start; p != NULL; p = p->next)
    printf("%d", p->info);
}
```

3) Insertion →

a) Insertion at start :→ If (start == NULL)

concept → start = new;



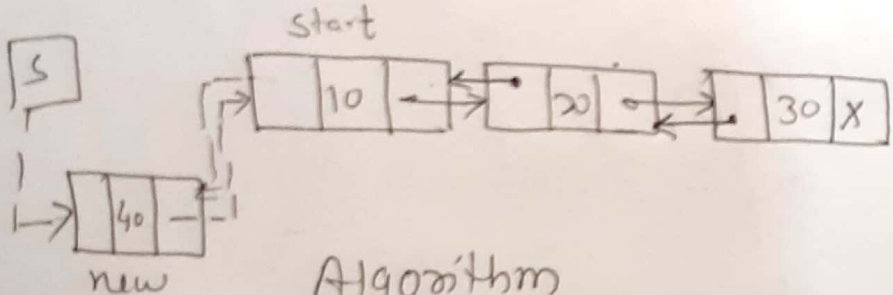


set new → next = start

start → prev = new

~~start = new~~

~~new~~ start = new



Algorithm

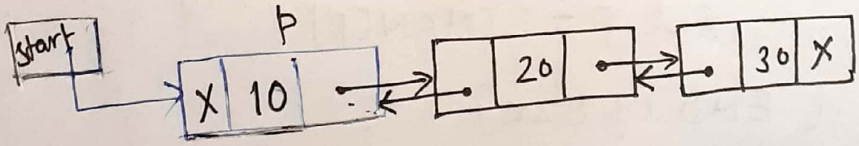
Insert_at_beg( )

1. new = AVAIL
2. set LINKP[new] = ~~AVAIL~~ NULL
3. Set LINKN[new] = NULL
4. INFO[new] = item
5. set LINKN[new] = start
6. set LINKP[start] = new
7. set start = new
8. Exit

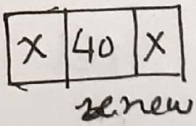
c function →

```
void insert_at_beg()
{
    struct node *new;
    new=(struct node *)malloc(sizeof(struct node));
    printf("Enter node info");
    scanf("%d", &new->info);
    new->prev=NULL;
    new->next=NULL;
    if(start==NULL)
        start=new;
    else
    {
        new->next=start;
        start->prev=new;
        start=new;
    }
}
```

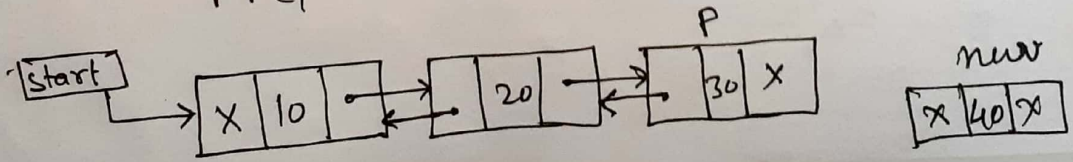
b) Insert at end -



if List is empty
start = new



else
move to last node
for(p=start; p->next!=NULL; p=p->next)



Pradeep Pant

(19)

o list is empty then

new → next = start

start = new

else

set p → next = new

new → prev = p

Algorithm →

gnsort_at_end()

1. new = AVAIL
2. INFO[new] = item
3. LINKP[new] = NULL
4. LINKN[new] = NULL
5. If (start = NULL) then

set start = new

[End of If]

else

for p = start

while (LINKN[P] != NULL)

set P = LINKN[P]

[END of else]

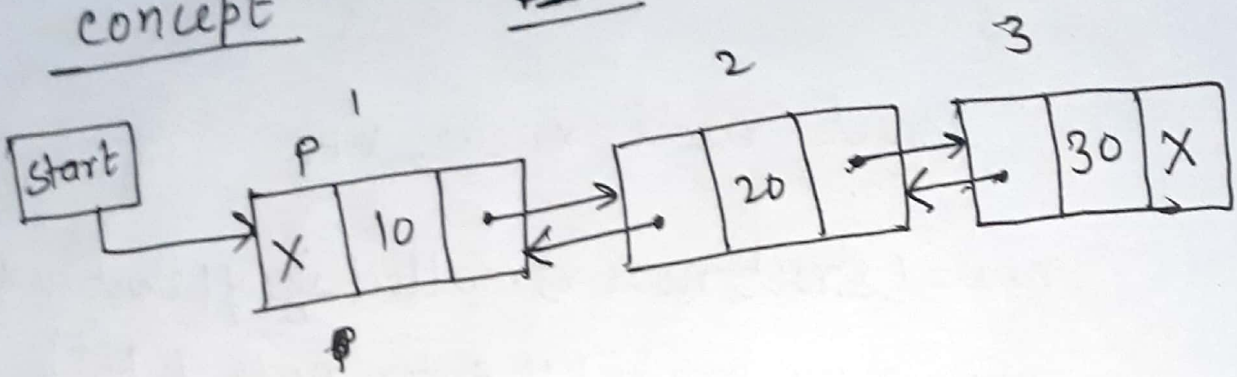
6. set LINKN[P] = new
7. LINKP[new] = P
8. Exit

c function →

```
void insert_at_end()
{
  struct node *new, *p;
new=(struct node *)malloc(sizeof(struct node));
new->po printf("Enter node info");
  scanf("%d", &node->info);
   new->prev = NULL;
   new->next = NULL;
   if(start==NULL)
      start=new;
   else
     {
      for(p=start; p->next!=NULL, p=p->next);
       p->next=new;
       new->prev=p;
      }
}
```

c) Insert after node→

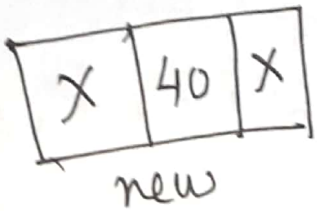
concept



if list is empty

start = new

else



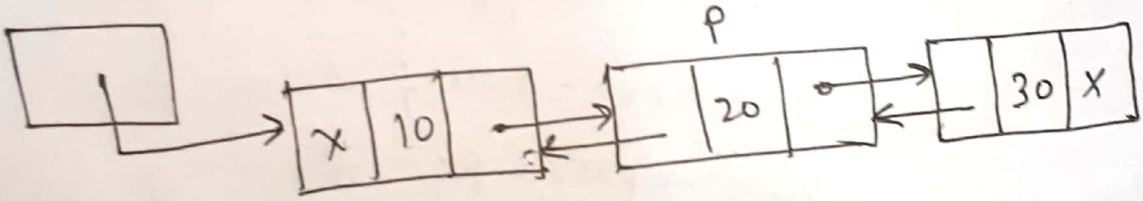
Input Node number after new node is inserted. (N)

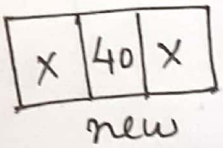
P = start

move pointer P to that node

```
for(i=1 to N-1)
    P = P→next
```

let N = 2





now set

new → next = P → next

P → next → prev = new

P → next = new

and new → prev = P

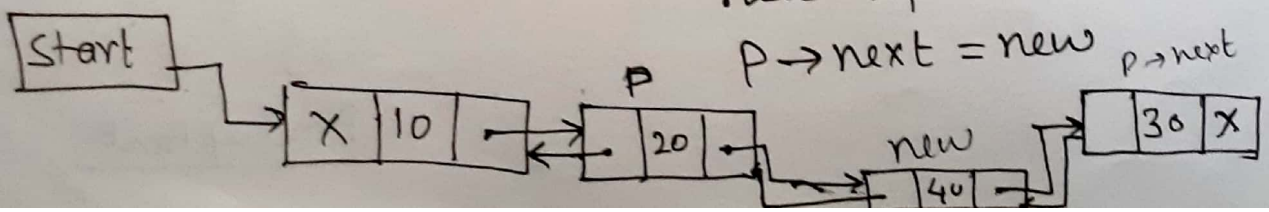
set

new → next = P → next

P → next → prev = new

new → prev = P

P → next = new



Algorithm →

Insert_afternode()

1. new = AVAIL
2. INFO[new] = item
3. LINKP[new] = NULL
4. LINKN[new] = NULL
5. if (start = NULL)
   start = new
   [end of if]
   else
   input N
   P = start
   for (i = 1 to N-1)
   P = LINKN[P]
   [END of loop]
   ~~set new->next = p->next~~
   ~~p->next~~ LINKN[new] = LINKN[P]
   LINKP[LINKN[P]] = new
   LINKP[new] = P
   LINKN[P] = new
   [END of else]
6. Exit

Pradeep Pant

Pradeep Pant

c function →

```
void insert_after_node()
{
 struct node *new, *P;
  int i, N;

new=(struct node *)malloc(sizeof(struct node));
new printf("Enter node info");
   scanf("%d", &node->info);
    new->prev= NULL;
    new->next= NULL;
 if(start==NULL)
    new= start;
 else
  {
   printf("Enter node number");
   scanf("%d", &N);
    ster P=start;
  for(i=1; i<N; i++)
      P= P->next;
    new->next= p->next;
 P->next->prev= new;
    new->prev= P;
    P->next= new;
 }
}
```

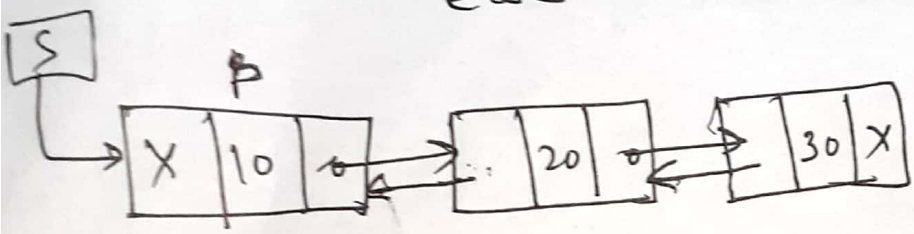
4) Deletion →

a) Delete first node.

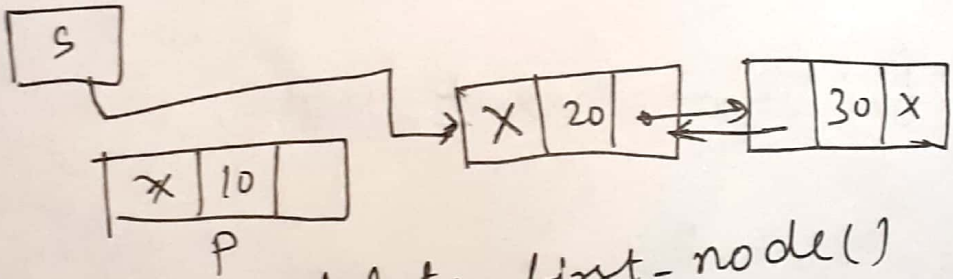
concept →

if (start == NULL)
print underflow

else



P = start
start = P → next
start → prev = NULL
free(P)



Algo → delete_first_node()

1. if start = NULL
   write "Underflow"
   [End of if]
   Exit.
2. p = start;
3. start = LINKN[P]
4. LINKP[start] = NULL
5. free(P)

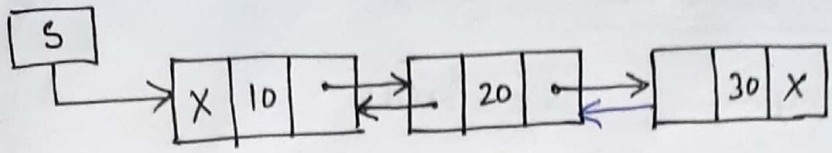
Pradeep Pant

c program →

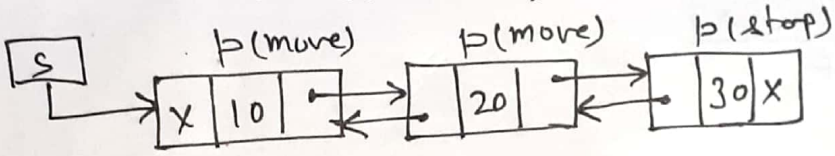
```
void delete_first_node()
{
  struct node *p;
   if (start == NULL)
     printf("Underflow");
   else
   {
      p = start;
     start = p->next;
     start->prev = NULL;
      free(p);
   }
}
else if (start->next == NULL)
   {
      p = start;
     start = NULL;
      free(p);
   }
```

b) Delete Last Node →

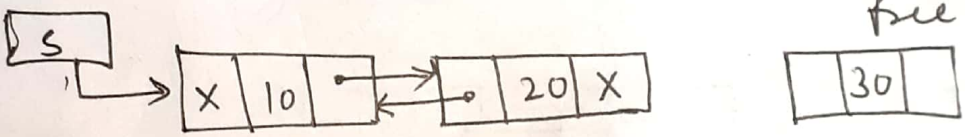
concept →



If List is empty, i.e. start == NULL
print underflow
else if (List contain only one node) i.e. start -> next == NULL
~~p = start~~ P = start start = NULL free(P)
else move P to last node
for (P = start; P -> next != NULL, P = P -> next);



now P -> prev -> next = NULL
free(P)



Algo → delete_last_node()

1. if start = NULL then
write "Underflow"
[END of if]
else if (~~delete~~ -> LINKN [start] = NULL)
~~start~~ -> P = start
start = NULL.
free(p)
else
~~for~~ P = start.
while (LINKN[P] != NULL)
P = LINKN[P]
[END of Loop]

LINKN[LINKP[P]] = NULL

free(p)

Exit.

c) Delete a particular Node

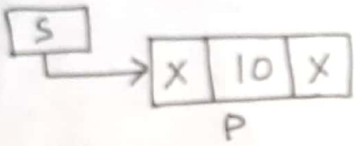
case1- List is empty $S == NULL$

start

NULL

print underflow

case2- List contains only one node $S \rightarrow next == NULL$

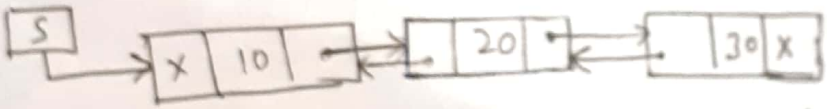


P = start

start = NULL

free(p)

case3- List contains 2 or more nodes



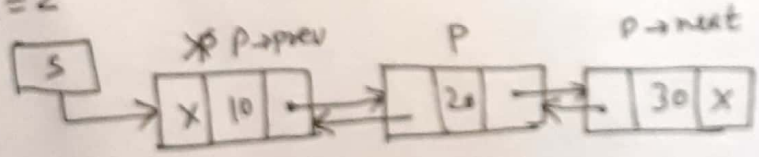
Input Node number to be deleted. (N)

P = start

for (i=1 to N-1)

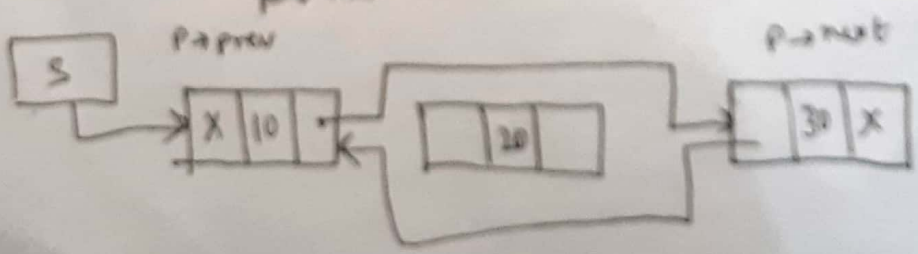
P = P → next

Let N=2



now $P \rightarrow next \rightarrow prev = P \rightarrow prev$

$P \rightarrow prev \rightarrow next = P \rightarrow next$



Case 4:- N is last node

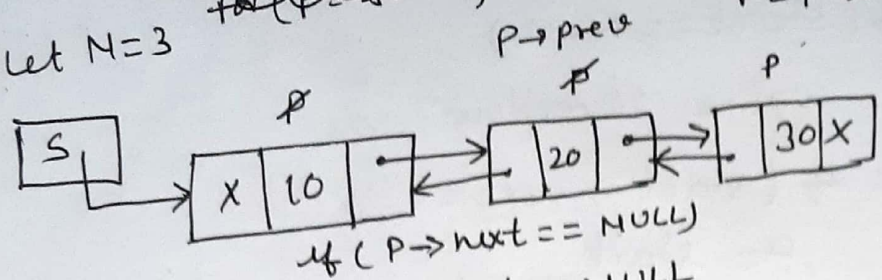
Let N=3

~~for(p=start;p~~

P = start

for (i=1 to N-1)

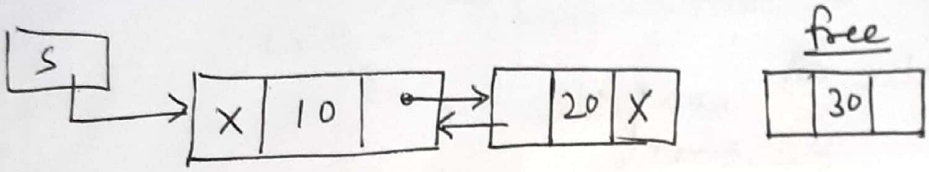
P = P → next



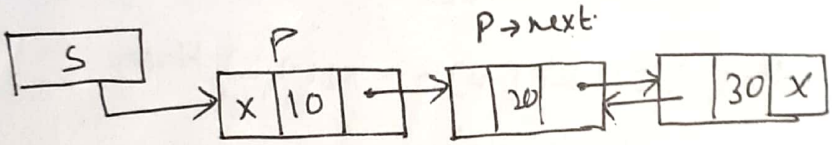
if (P → next == NULL)

P → prev → next = NULL

free(P)



Case 5 → N = 1



P = start

for (~~P=start~~ i=1 to N-1)

P = P → start

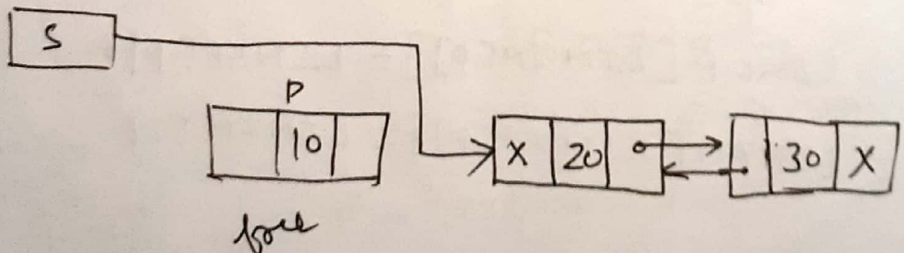
if (P → prev == NULL)

~~start~~

start = P → next

P → next → prev = NULL

free(P)



Algo → delete_particular_node()

```
if (start == NULL)
    write "overflow"
else if (start -> next == NULL)
    p = start
    start = NULL
    free(p)
else
    input N
    p = start
    for (i = 1 to N-1)
        p = LINKN[p]
    if (LINKP[p] == NULL) then
        p = start
        start = LINKN[p]
        LINKP[LINKN[p]] = NULL
        free(p)
    else if (LINKN[p] == NULL)
        LINKN[LINKP[p]] = NULL
        free(p)
    else
        LINKP[LINKN[p]] = LINKP[p]
        LINKN[LINKP[p]] = LINKN[p]
exit.
```

c functions →

```
void delete_perticular_node()
{
  struct node *p;
  int i, N;
  if (start == NULL)
    printf("Underflow");
  else if (start->next == NULL)
    {
      p = start;
      start = NULL;
      free(p);
    }
  else
    {
      printf("Enter Node number");
      scanf("%d", &N);
      p = start;
      for (i=1; i<N; i++)
        p = p->next;
      if (p->prev == NULL)
        {
          p = start;
          start = p->next;
          p->next->prev = NULL;
          free(p);
        }
      else if (p->next == NULL)
        {
          p->prev->next = NULL;
          free(p);
        }
```

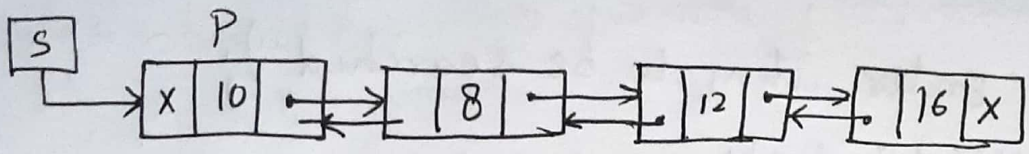
```
else
 {
    p->next->prev = p->prev;
    p->prev->next = p->next;
      free(p);
  }
 }
}
```

** C function (Delete last Node)

```
void delete_last_node( )
{
   struct node *p;
   if (start == NULL)
      printf("underflow");
   else if (start->next == NULL)
    {
       p = start;
      start = NULL;
       free (p);
    }
   else {
     for (p = start; p->next != NULL; p=p->next);
          p->prev->next == NULL;
            free(p); } }
```

5) Searching → (both for sorted or unsorted list) (7)

concept →



e = 12 e = 24

Input item to be searched (e)

for(P = start; P != NULL && P->info != NULL; P = P->next);

if (P == NULL)
not found
else
found

Algo → search()

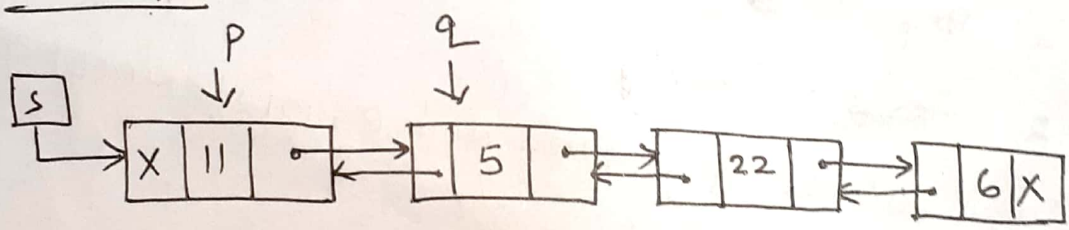
1. Input e
2. P = start;
3. while (P != NULL and P->info != NULL)
4. P = P->next.
[End of loop]
5. if (P == NULL)
write "Not found"
else
write "Found"
6. Exit

c function

```
void search()
{ int e;
 struct node *p;
 printf("Enter item to be searched");
 scanf("%d", &e);
 for(p= start; p!=NULL && p->info!= e; p=p->next);
   if(p==NULL)
      printf("not found");
   else
      printf("Found");
}
```

6) sorting →

concept →

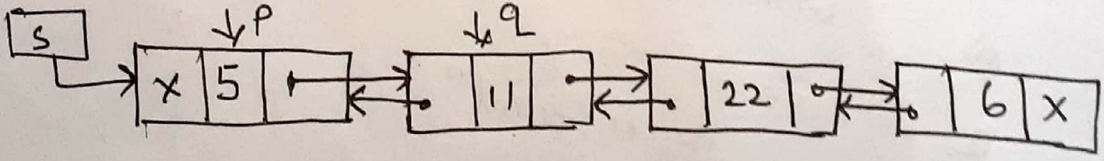


p = start

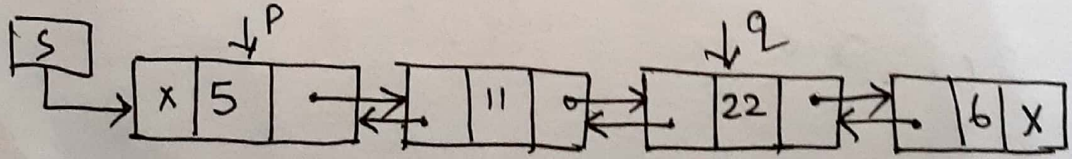
q = p->next

check if(p->info > q->info) 11 > 5 → T

swap(p->info, q->info)



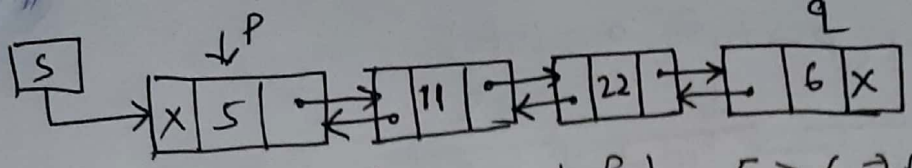
now update q = q->next



again check if(p->info > q->info) 5 > 22 T

no swap

now update q = q → next

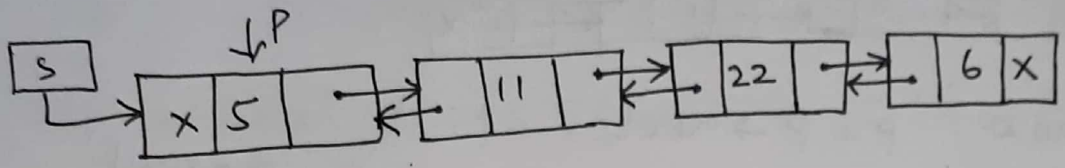


now check if(p → info > q → info) 5 > 6 ⇒ F

no swap

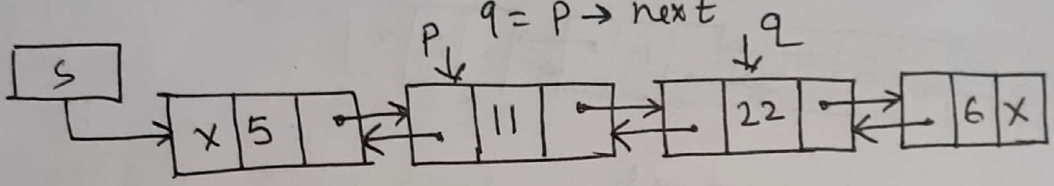
now update q = q → next

q = null



now update p = p → next

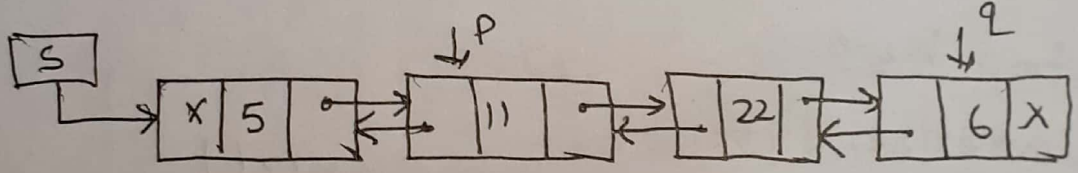
q = p → next



if(p → info > q → info) 11 > 22 ⇒ F

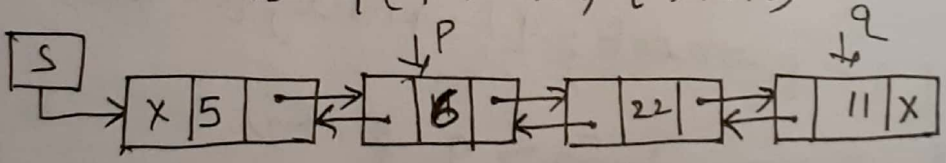
no swap

now update q = q → next

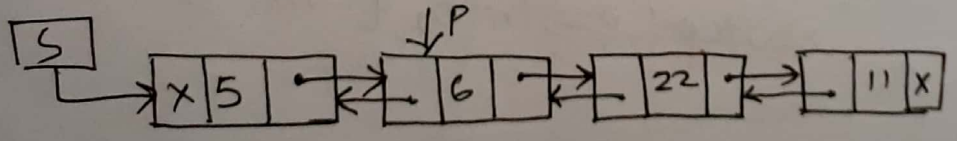


if(p → info > q → info) 11 > 6 ⇒ T

swap(p → info, q → info)



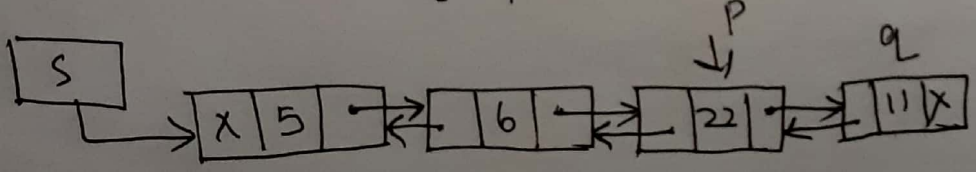
now q = q → next



q = null

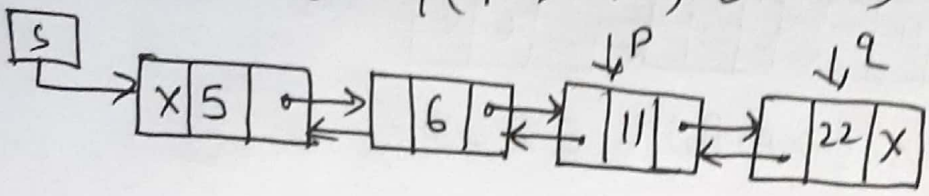
now p = p → next

q = p → next

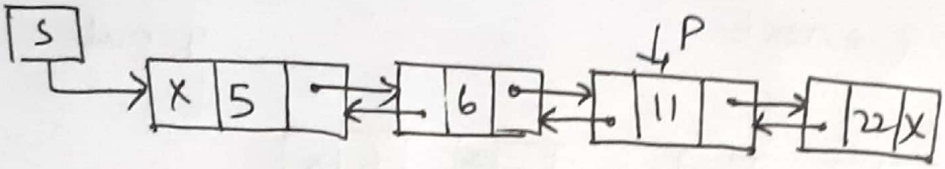


if (p->info > q->info) 22 > 11 ⇒ T

swap(p->info, q->info)

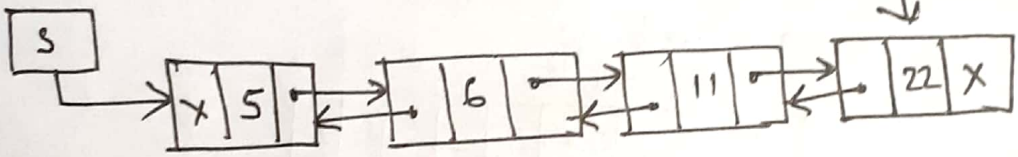


now q = q->next



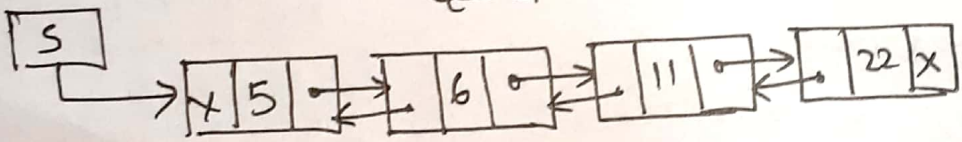
now p = p->next

q = p->next



now p = p->next

~~q = p->next~~



stop here

To summerize,

```
for ( p = start ; p != NULL; p = p->next)
   {
      for ( q = p->next ; q != NULL; q = q->next)
         {  if ( p->info > q->info)
               swap ( p->info, q->info)
         }
   }
```

Algo →

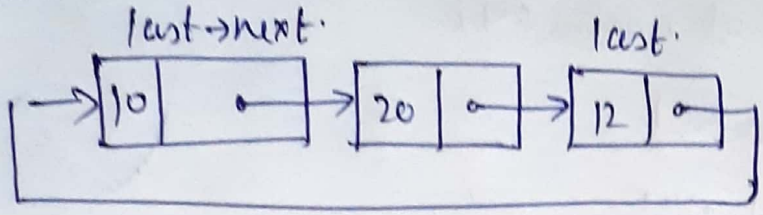
sort()

1. ~~for~~ p = start
2. while (p != NULL)
3. q = ~~p->next~~ LINKN[P]
4. repeat steps 5 & 6 while (q != NULL)
5. if (~~p->~~ INFO[P] > INFO[q]) then
   swap(~~p->~~ INFO[P], INFO[q])
   [End of if]
6. q = LINKN[q]
   [END of step 4 loop]
7. P = LINK[P]
   [END of step 2 loop]
8. Exit

C function →

```
void swap()
{
struct node *p, *q;
   int t;
for (p = start; p != NULL; p = p->next)
    for (q = p->next; q != NULL; q = q->next)
        if (p->info > q->info)
        { t = p->info;
          p->info = q->info;
        ; q->info = t;
}
```

Circular Linked List (Singly)



start = last → next

Operation on Circular List

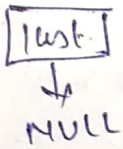
1) Create → Define structure

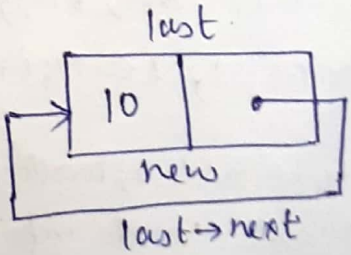
```
struct node.
{
  int info;
struct node *next;
} *last = NULL;
```

Concept →

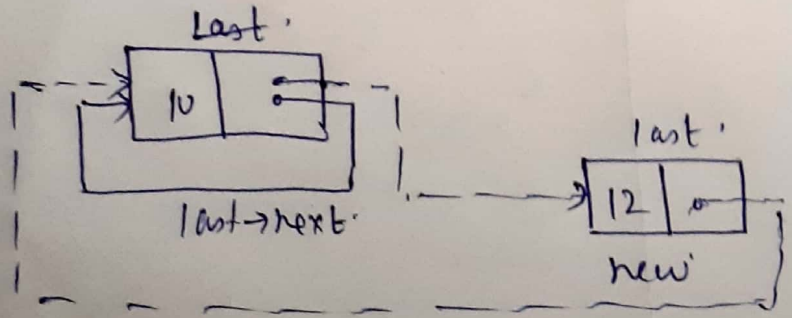
```
Input NO. of nodes (N)
for (i=1 to N)
new = malloc()
new → info = item
if (last == NULL)
{
  last = new
  new → next = new
}
```

i=1





i=2




```
new → next = last → next
last → next = new
last = new
```

Algorithm → create-circular-list()

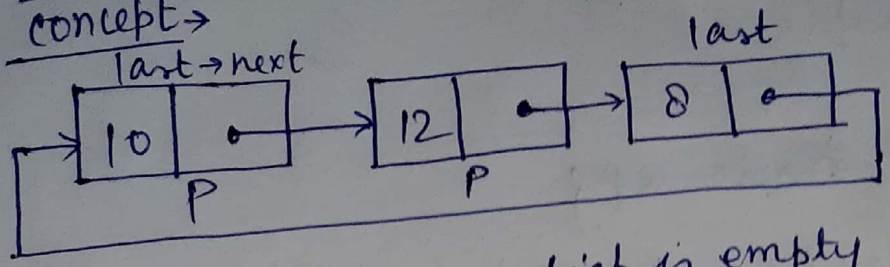
1. Input No. of nodes (N)
2. for (i=1 to N)
3. new = AVAIL
4. INFO[new] = item
5. if (last == NULL) then
   last = new
   LINK[new] = new
   [END of if]
   else
   LINK[new] = LINK[last]
   LINK[last] = new
   last = new
   [END of else]
   [END of step2 loop]
6. Exit.

C function →

```
void create_circular_list()
{ int i, N;
  struct node *new;
  printf("Enter No. of nodes");
  scanf("%d", &N);
  for(i=1; i<=N; i++)
  {
new=(struct node *) malloc(sizeof(struct node));
printf("Enter node info");
scanf("%d", &new->info);
if(last==NULL)
 { last=new;
new->next=new;
 last=new;
 }
else
 {
new->next=last->next;
 last->next=new;
 last=new;
 }
}
```

display or traverse →

concept →



if (last == NULL) then list is empty otherwise.

set P = last → next

print P → info

set P = P → next.

update P = P → next until P != last → next.

Algorithm →

traverse()

1. if (last == NULL) then
2. write "List is empty"
   exit
2. P = LINK[last]
3. write INFO[P]
4. while (P != last) P = LINK[P]
5. while (P != LINK[last])
6. write INFO[P]
7. P = LINK[P]
8. [END of step 2 loop]
8. Exit.

c function →

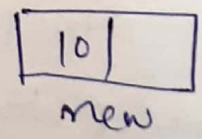
```
void display()
{
  struct node *p;
  if (last == NULL)
    printf("List is empty");
  else
    {
      p = last->next;
      do
      {
        printf("%d", p->info);
        p = p->next;
      } while(p != last->next);
    }
```

3) Insertion →

a) At begining -

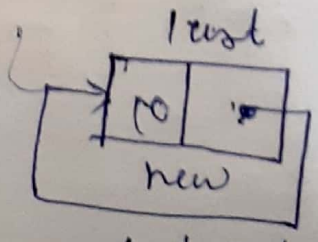
Concept

1) List is empty



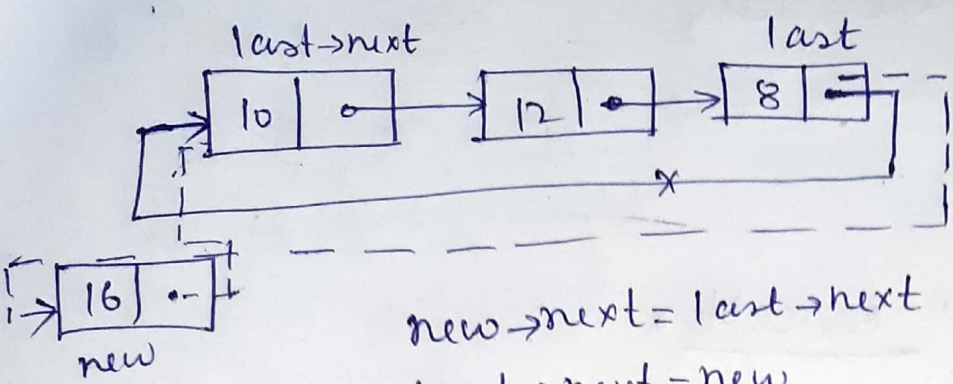
last = new

new → next = new



3

2) List is not empty



new->next = last->next

last->next = new

Algo→ Insert_at_begining()

1. new = AVAIL
2. INFO[new] = item
3. if (last == NULL) then
   last = new
   LINK[new] = new
   else
   LINK[new] = LINK[last]
   LINK[last] = new
4. Exit

function→

```
void insert_at_beg()
{
  struct node *new;
new=(struct node *)malloc(sizeof(struct node));
 printf("Enter node info");
 scanf("%d", &new->info);
  if(last == NULL)
  { last = new;
  new->next = new;
  }
else
 {
```

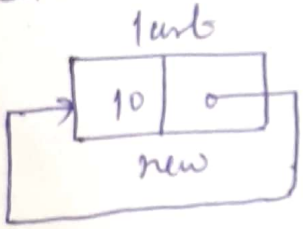
```
        new->next = last->next;
        last->next = new;
    }
}
```

b) Insertion at last → concept →

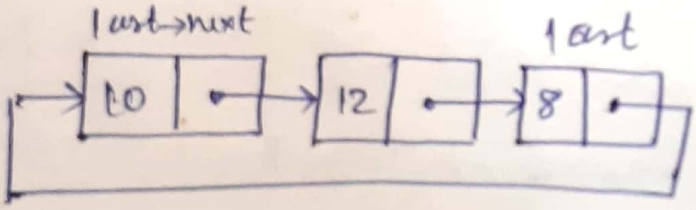
1) List is empty

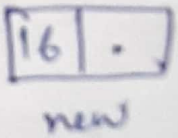
last = new

new->next = new



2) List is not empty

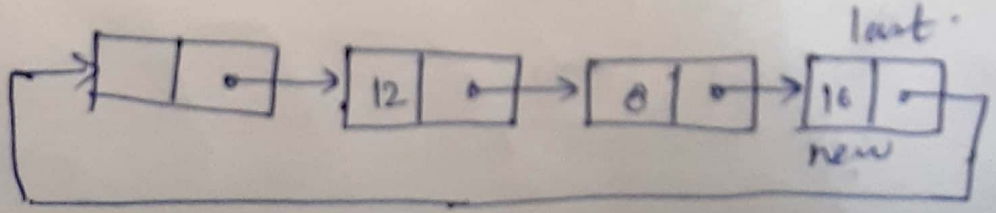




new->next = last->next

last->next = new

last = new



(c) Insertion after node →

(d) Algorithm →

Insertion_at_beg( )

1. if (last == NULL)
   last = new
   new → next
   Insertion_at_beg( )

1. new = AVAIL
2. INFO[new] = item
3. if (last == NULL) then
   last = new
   new → next = new
   [End of if]
   else
   new → next = last → next
   new → next = new
   last = new
   [End of else]
4. Exit

Functions →

```
Void insert_at_beg()
{
  struct node *new;
new = (struct node *) malloc(sizeof(struct node));
printf("Enter node into");
scanf("%d", &new→info);
```

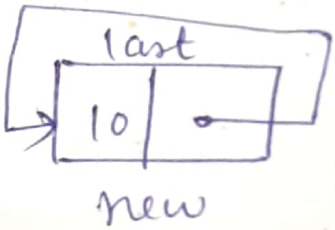
```
if (last == NULL)
{
    last = new;
new -> next = new;
}
else
{
    new -> next = last -> next;
    last -> next = new;
    last = new;
}
```

c) Insertion after a node →

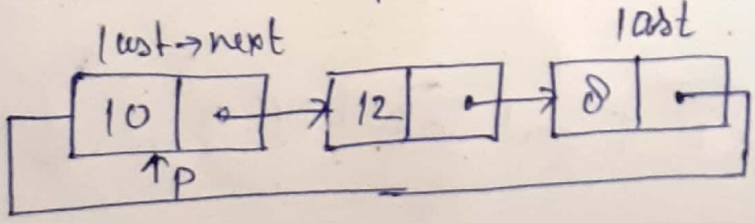
concept →

1) List is empty

```
if (last == NULL)
{
  last = new
new -> next = new;
}
```

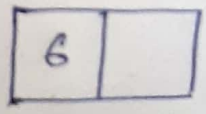


2) List is not empty



~~new->next = last->next~~

```
InputN(3)  p = last->next
for (i=1 to N-1)
    p = p->next
```



```
new = malloc()
new → info = item

new → next = p → next
p → next = new          /// last node

if (p → next == last → next)
{
  new → next = last → next;
  last → next = new;
    last = new;
}
else
{
  new → next = p → next
  p → next = new
}
```

Algo → Insert_after_node()

1. Input Node Number(N)
2. for [i = 1 to N-1)
3. P = P[LINK[P]
   [End of step 2 loop]
4. if (last == NULL)
   last = new
   new[LINK[new]] = new
   [END of if]
   else

Algo→ Insert_after_node()

1. Input N
2.
3. new = AVAIL
4. INFO[new] = item
5. if (last == NULL)
6. Last = new
7. LINK[new] = new
8. [END of if]

else

P = Link[last]

for (i = 1 to N-1)

P = LINK[P]

if (LINK[P] = LINK[last])

LINK[new] = LINK[last]

LINK[last] = new

Last = new

else.

LINK[new] = LINK[P]

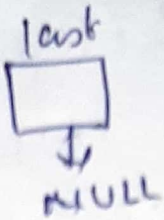
LINK[P] = new

6. exit.

4) Deletion →

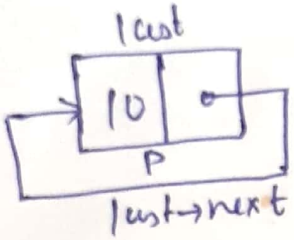
a) Delete first Node –

case1 → List is empty



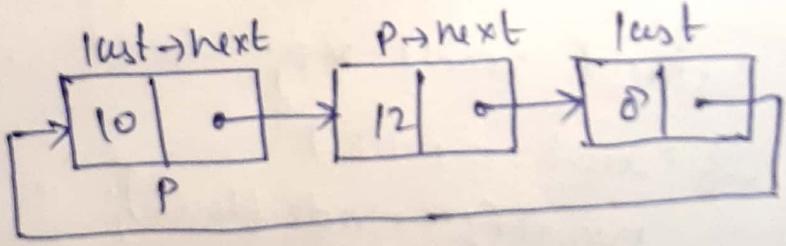

```
if(last == NULL)
  printf("Underflow");
```

case2 → List contains only one node




```
if(last→next==last)
{
  p=last→next
  free(p)
  last→next=NULL
}
```

case 3 – List contains more than one nodes




```
p=last→next
last→next=p→next
free(p)
```

Algo → delete_first_node( )

```
if (last == NULL)
    write "Underflow"
[END of if]
else if ( LINK[last] == last)
    p = LINK[last]
    free(p)
    LINK[last] = NULL
else
    p = last->next
    LINK[last] = LINK[p]
    free(p)
```

C function →

```
void delete_first_node( )
{
    struct node *p;
    if (last == NULL)
        printf("Underflow");
    else if (last->next == last)
    {
        p = last->next;
        free(p);
        last->next = NULL;
    }
    else
    {
        p = last->next;
        last->next = p->next;
        free(p);
    }
```

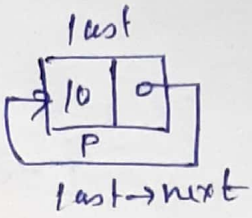
2) Delete last node

concept

case 1- List is empty

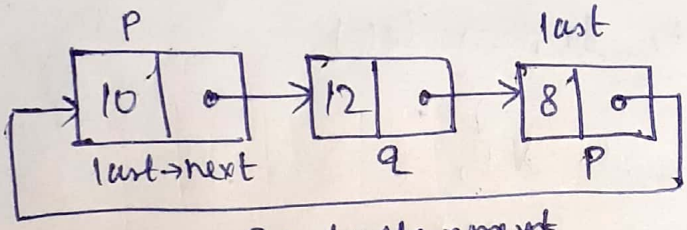
```
if(last==NULL)
    printf("Underflow")
```

case 2:- List contains only one node




```
if(last→next==last)
{
    p=last→next;
    free(p);
    last→next=NULL;
}
```

case 3- List contains more than one node



~~p=last→next~~

```
for(p=last→next; p→next!=last→next; p=p→next)
    q=p
q→next=last→next
    free(p)
```

Algo→

```
Delete-first-node()
    if(last==NULL)
        write "Underflow"
    else if( LINK[last]==last)
        P=LINK[next]
        free(p)
        LINK[last]=NULL
```

else

~~for(P=last->next;~~ P=LINK[last]

while(LINK[P] != LINK[last])

q=P

P=LINK[P]

[END of loop]

LINK[q]=LINK[last]

free(P)

exit.

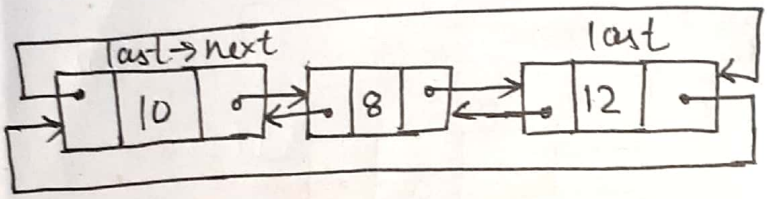
C function →

```
void delete_last_node()
{
  struct node *P;
  if(last==NULL)
    printf("Underflow");
  else if(last->next==last)
    {
      p=last->next;
      free(P);
      last->next=NULL;
    }
  else
    {
  for(P=last->next; P->next!=last->next; P=P->next)
      q=P;
    q->next=last->next;
      free(P);
    }
```

Double circular Linked List:-

Structure Definition:-

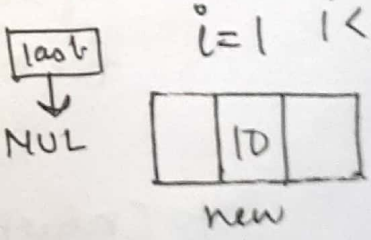
```
struct node
{
  int info;
  struct node *prev;
  struct node *next;
} *last= NULL;
```

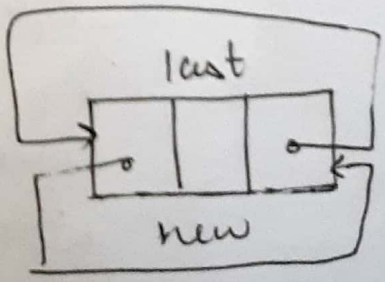


Create→

concept→

input No. of nodes (N)
for (i=1 to N)
{
  new = malloc()
scanf("%d", &new->info);
if list is empty
i.e. if (last == NULL)
{
  last = new;
new->next = last->next
new->prev = last->next;
}





{

new→next = last→next

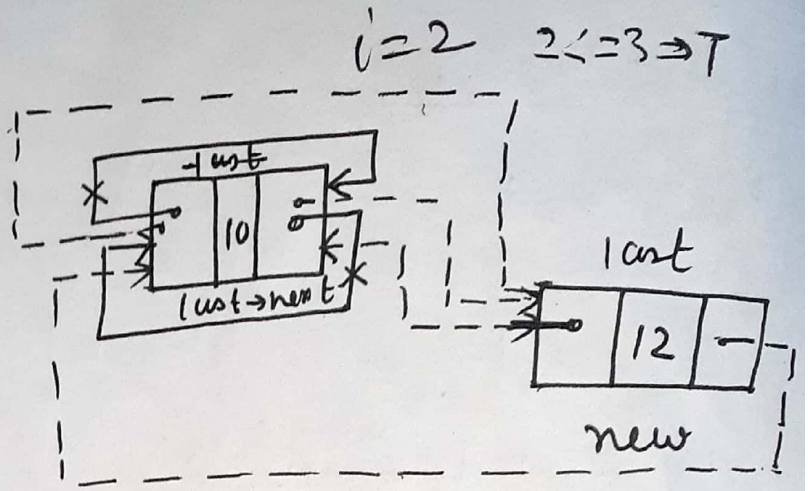
new→prev = last

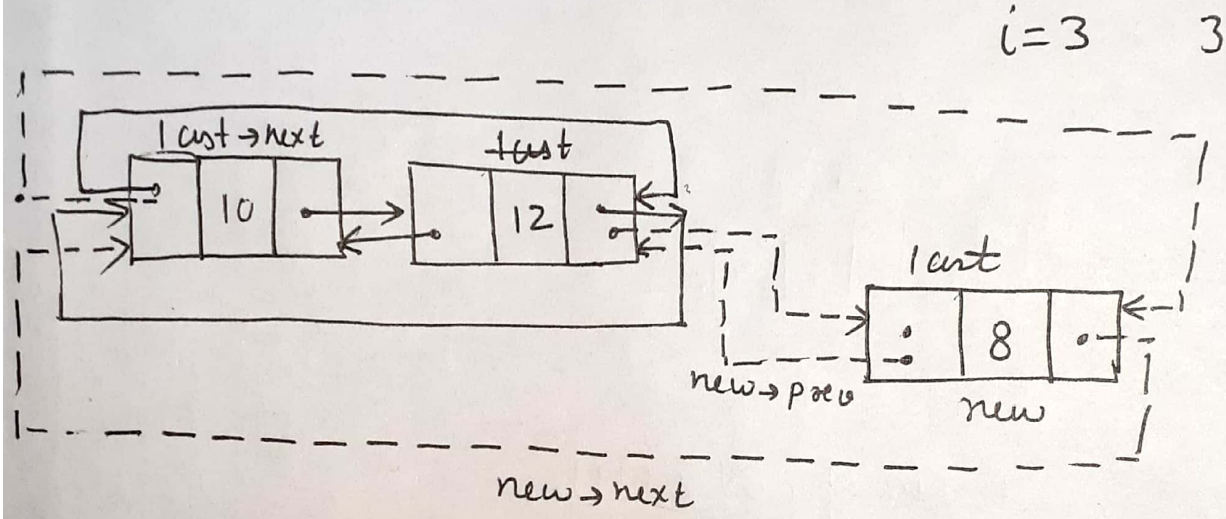
last→next→prev = new

last→next = new

last = new

}





i=4 4<=3 ⇒F

Algorithm → Create_doubly_circular()

Input Number of nodes (N)

for (i=1 to N)

new = AVAIL

Info[new] = item

if (last = NULL)

last = new

LINKN[new] = LINKN[last]

LINKP[new] = last

[END of if]

else

LINKN[new] = LINKN[last]

LINKP[new] = last

LINKP[LINKN[last]] = last

LINKN[last] = new

last = new

[End of else]

[END of loop]

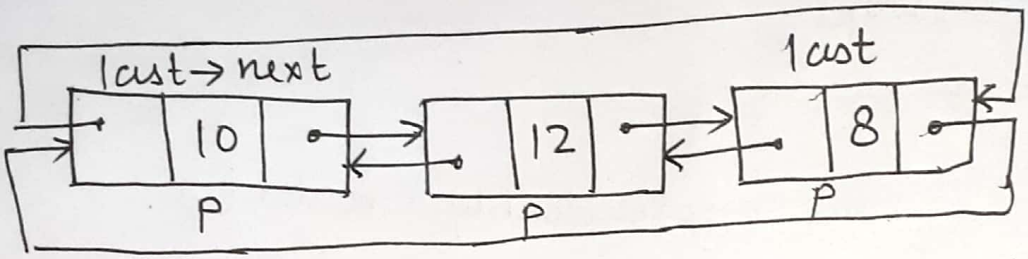
Exit

c function—

```
void create_doubly_circular()
{
 struct node *new;
  int i, N;
  printf("Enter no. of nodes");
  scanf("%d", &N);
   for(i=1; i<=N; i++)
    {
    new=(struct node *)malloc(sizeof(struct node));
    printf("Enter node info");
    scanf("%d", &new->info);
     if(last==NULL)
      {
        last=new;
      new->next=last->next;
      new->prev=last;
       }
    else
      {
       new->next=last->next
       new->prev=last
       last->next->prev=new
       last->next=new
```

```
        last = new;
      }
    }
  }
```

2) Traversing →



concept → ~~set p to first node, i.e. p = last → next~~

first check list is empty or not

if (last == NULL) then print list is empty

otherwise

set p = last → next

do

{

process (P)

set p = p → next

} while (P != last → next)

Algo →

```
Traverse_list()
  If (last == NULL)
     write "List is empty"
  else
       p = last → LINKN (next)
    do
     {
       write Process (P)
        P = LINKN (next)
     } while (P != last → LINKN (next))
```

c function →

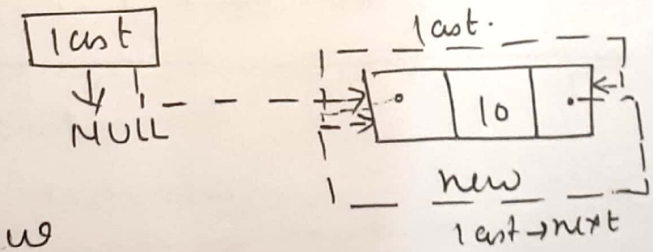
```
void display()
{
  struct node *p;
  if(last == NULL)
    printf("List is empty");
  else
    {
      do
        {
          printf("%d", p->info);
          p = p->next;
        } while(p != last->next);
    }
}
```

) Insertion →

a) Insert a node at begining.

concept

Case1: List is empty



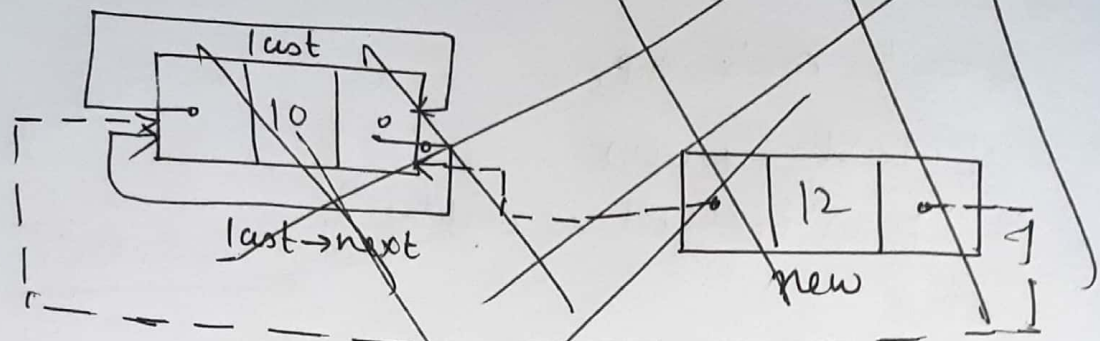
last = new

new→next = last→next

new→prev = last

(6)

Case 2 — List ~~contains adjacent node~~ is not empty



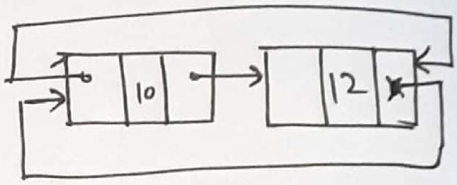
new → next = last → next

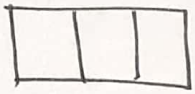
new → prev = last

~~last → next = prev~~

last → next → prev = new

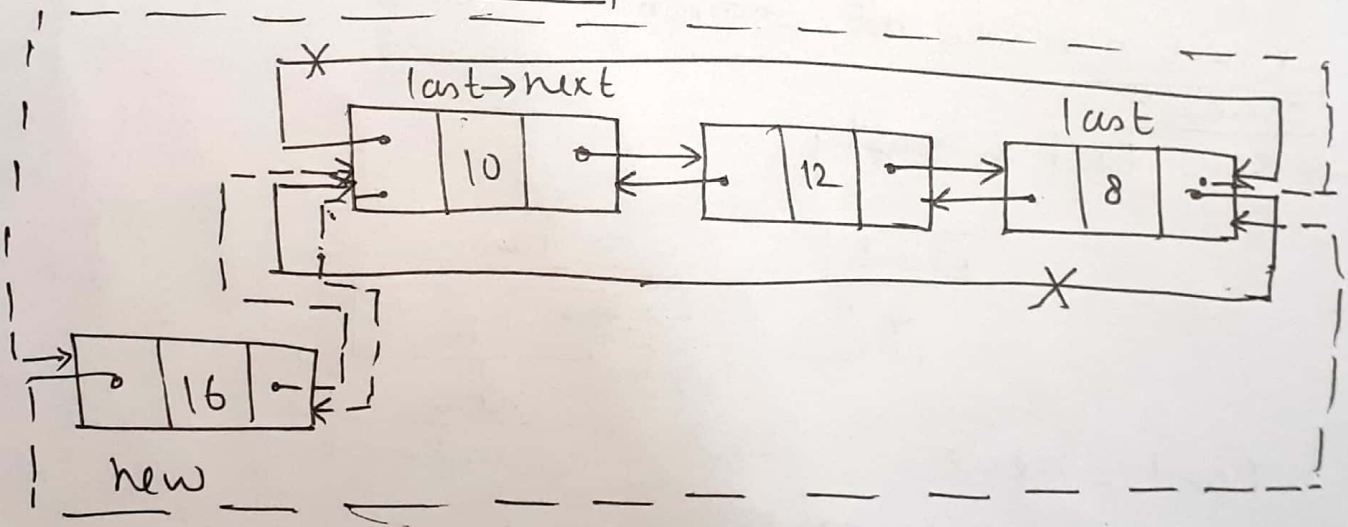
last → next =





---

list is not empty



new → next = last → next

new → prev = last

last → next → prev = new

last → next = new;

Algorithm

insert_at_beg()

new = AVAIL

new → info = item

if (last == NULL)

last = new

new → next = last → next

new → prev = last

[End of if]

else

new → next = last → next

new → prev = last

last → next → prev = new

last → next = new

Exit.

if (last == NULL)
last = new
LINKN[new] = LINKN[last]
LINKP[new] = last

LINKN[new] = LINKN[last]
LINKP[new] = last
LINKP[LINKN[last]] = new
LINKN[last] = new

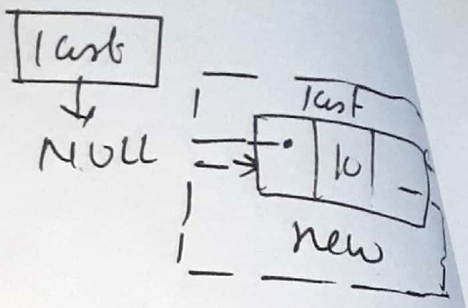
c function

```
void insert_at_beg()
{
struct node *new;
new = (struct node *) malloc (sizeof (struct node));
 printf("Enter node info");
 scanf("%d", &new->info);
  if (start == NULL)
   {
      last = new;
    new->next = last->next
    new->prev = last
   }
  else
    {
      new->next = last->next;
      new->prev = last;
last->next->prev = new;
      last->prev = new;
    }
}
```

b) Insert at last →

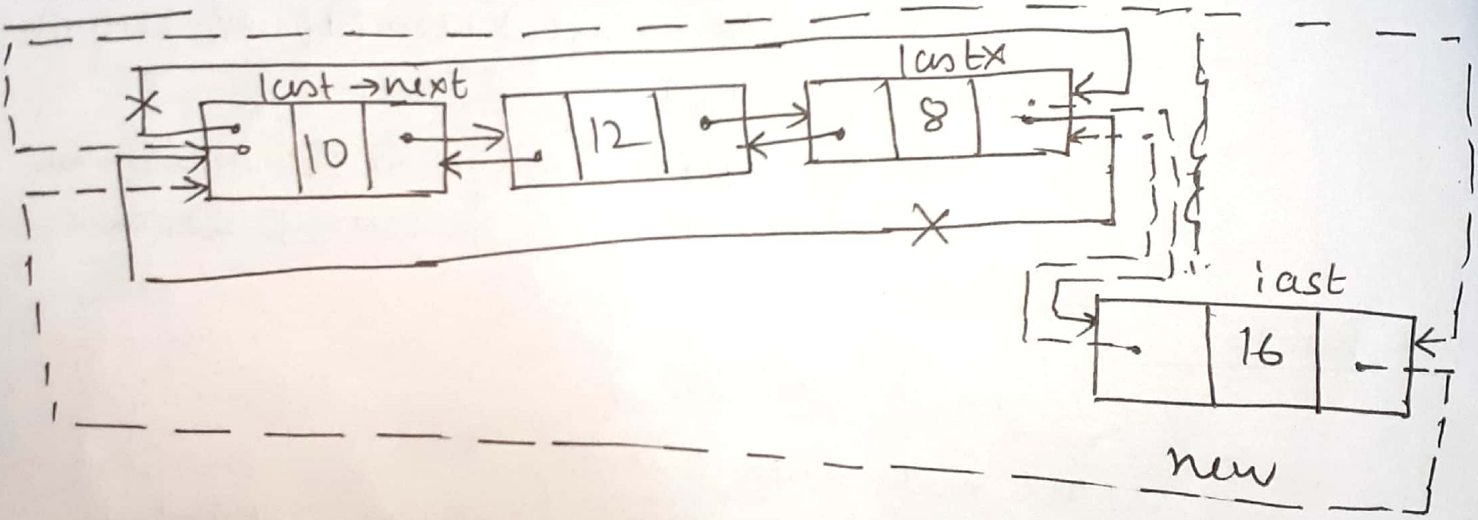
concept →

case 1 → List is empty




```
if (last == NULL)
{
    last = new
new → next = last → next
new → prev = last
}
```

case 2 → List is not empty




```
new → next = last → next
new → prev = last
last → next → prev = new
last → next = new
last = new
```

Algorithm →

insert_at_beg()

new = AVAIL

INFO(new) = item

if (last == NULL)

last = new

LINKN[new] = LINKN[last]

LINKP[new] = last

[End of if]

else

LINKN[new] = LINKN[next]

LINKP[new] = last

LINKP[LINKN[last] = new

LINKN[last] = new

last = new

[END of else]

Exit

C function →

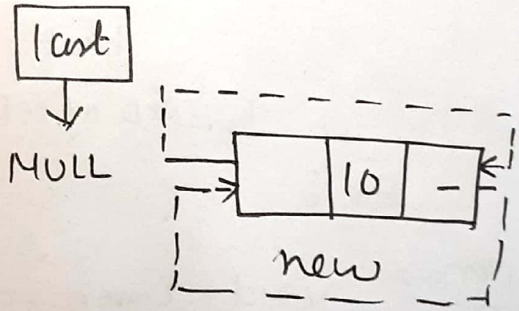
```
void insert_at_beg()
{
 struct node *new;
new = (struct node *) malloc(sizeof(struct node));
 printf("Enter node info");
 scanf("%d", &new->info);
  if (last == NULL)
  {
      last = new;
   new->next = last->next;
```

```
        new->prev=last;
    }
    else
    {
        new->next=last->next;
        new->prev=last;
    last->next->prev=new;
        last->next=new;
        last=new;
    }
}
```

c) Insert after node

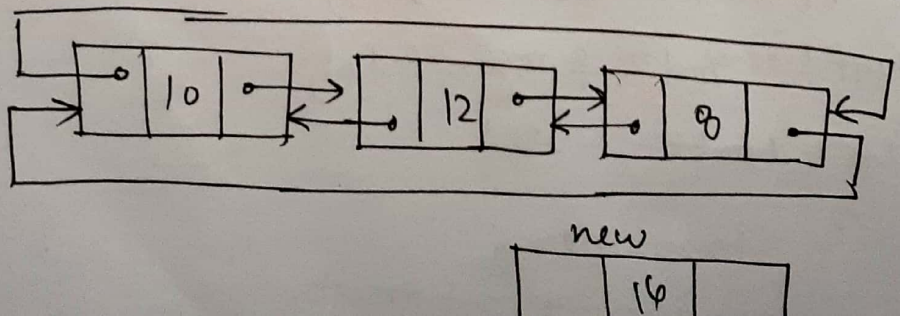
Concept→

Case1→ List is empty




```
if(last==NULL)
{
    last=new
    new->next=last->next
    new->prev=last
}
```

Case2→ List is not empty



p = last → next

enter node Number (N)

for (i = 1 to N-1)

p = p → next

case a) if p points to last node

if (p → next == last → next)

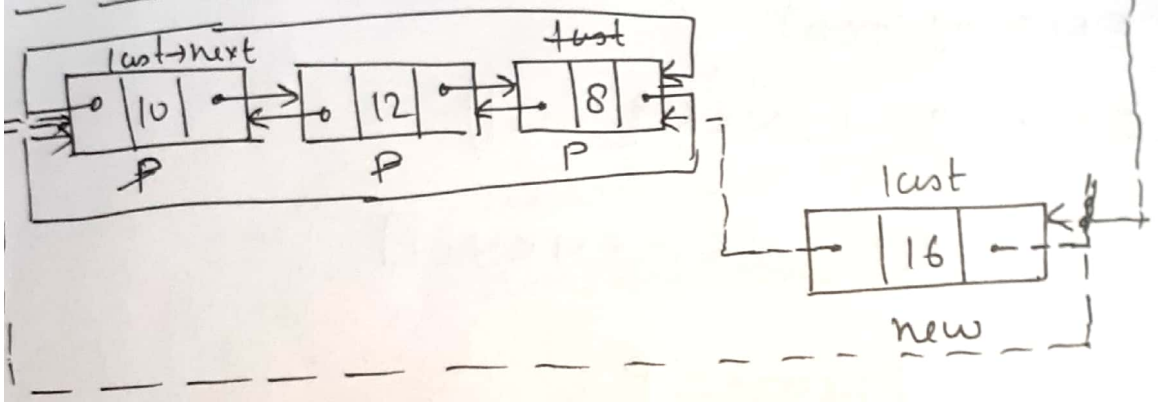
{ new → next = ~~p → next~~ last → next

new → prev = p

p → next → prev = new

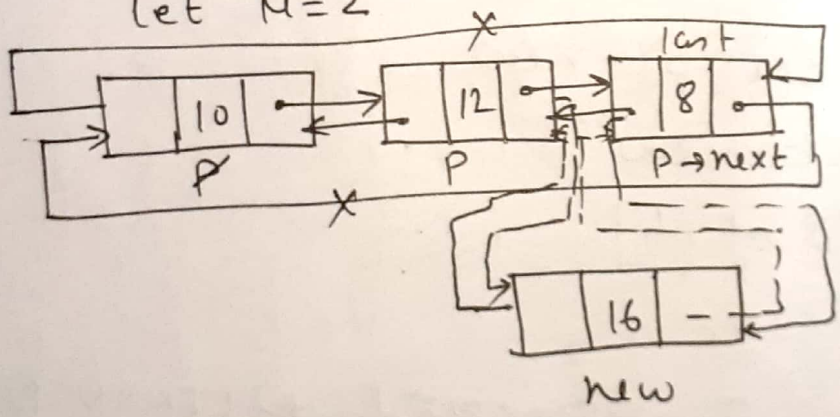
p → next = new last = new

let N = 3



case b) p is not a last node.

let N = 2



↳ new → next = p → next

new → prev = p

p → next → prev = new

p → next = new

Algorithm

Insert_after_node

new = AVAIL

INFO[new] = item

input node number (N)

p = LINKN[last]

for(i = 1 to N-1)

p = LINKN[P]

[END of loop]

if (LINKN[P] = LINKN[last])

LINKN[new] = LINKN[P]

LINKP[new] = P

LINKP[LINKN[P]] = new

LINKN[P] = new

last = new

[END of if]

else

LINKN[new] = LINKN[P]

LINKN[new] = P

LINKP[LINKN[P]] = new

LINKN[P] = new

[END of else]

Exit.

e...function →

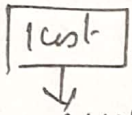
```
void insert_after_node()
{
  int i, N;
  struct node *new, *p;
new=(struct node *) malloc (sizeof (struct node));
  printf("Enter node info");
  scanf("%d", &new->info");
  if(last==NULL)
    {
      last=new;
    new->next=last->next
    new->prev=last
     }
else
  {
    p=last->next;
  printf("Enter node number");
   scanf("%d", &N);
   for(i=1; i<N; i++)
       p=p->next;
   if(p->next==last->next)
    {
      new->next=p->next;
      new->prev=p;
   p->next->prev=new;
      p->next=new;
      last=new;
    }
```

```
        else
        {
            new->next= p->next;
            new->prev= p;
        p->next->prev = new;
          p->next= new
        }
    }
}
```

4) Deletion →

a) Delete first node →

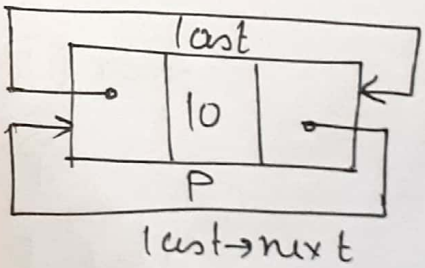
Case 1 → List is empty



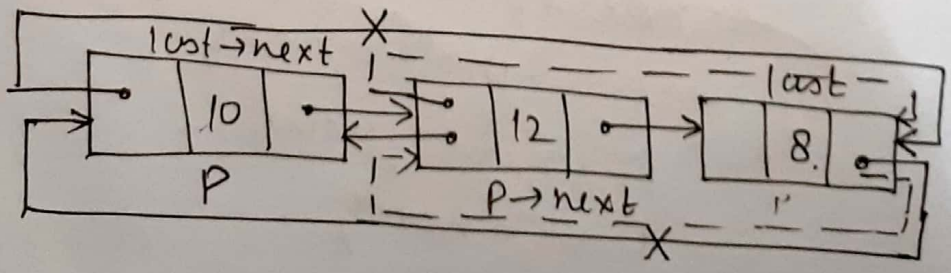

```
if (last == NULL)
  printf("~~overflo~~ underflow");
```

Case 2 → List contains only one node

```
p= last->next
if ( p->next== last->next)
  {
    free (p);
   last = NULL;
  }
```



Case 3 → List contains more than one node



last→next = p→next;
p→next→prev = last
free(p)

Algorithm→

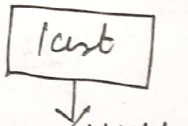
Delete_first_node()
if (last == NULL)
write printf "Underflow");
[End of if]
else
p = LINKN[last]
if (LINKN[P] == LINKN[last])
free(P)
Last = NULL
[END of if]
else
LINKN[last] = LINKN[P]
LINKP[LINKN[P]] = last
free(P)
[End of else]
[End of else]
Exit.

C function→

```
void delete_first_node()
{
    struct node *p;
    if(last == NULL)
        printf("Underflow");
    else
    {
        P = last→next;
```

```
if (p->next == last->next)
{
    free(p);
    last = NULL;
}
else
{
    last->next = p->next;
    p->next->prev = last;
    free(p);
}
}
}
```

b) Delete last node→

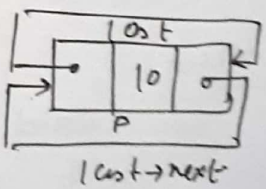


case1→ if list is NULL

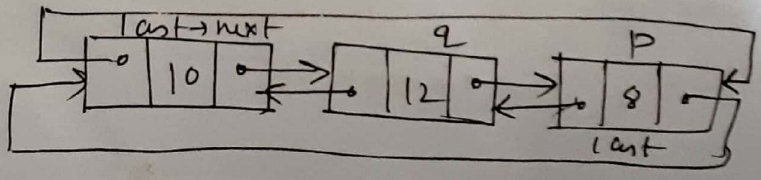
```
if (list == NULL)
  printf("Underflow");
```

case2→ List contain only one node

```
p = last->next
if (p->next == last->next)
{
    free(p);
    last = NULL
}
```



case3 List contains more than one node.



```
p=last
for(q=last->next; q->next!=last; q=q->next);
    q->next=last->next;
last->next->prev=q;
    last=q;
free(p);
```

Algorithm →

```
delete_last_node()
if(last==NULL)
    write "Underflow"
else
    p=last LINKN[last]
if(LINKN[P]==LINKN[last])
        free(P)
        last=NULL
        [End of if]
else
    p=last
for(q=last->next; LINKN[q]!=last]
while(q->next!=P)
  q=q->next
  [END of loop]
while(LINKM[q]!,=P]
    q=LINKN[q]
  [END of loop]
LINKM[q]=LINKN[last]
LINKP[LINKM[last]]=q;
```

last = q

free(p)

[END of else]

exit.

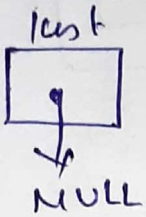
C function →

```
void delete_last_node()
{
  struct node *p, *q;
     p=last->next;
  if(last== NULL)
     printf("Underflow");
  else if(p->next==last->next)
    {
      free(p);
      last=NULL;
    }
  else
    {
      p=last;
    for(q=last->next; q->next!=p; q=q->next);
       q->next=last->next;
    last->next->prev=q;
       last=q;
       free(p);
    }
}
```

c) Delete a particular node.-

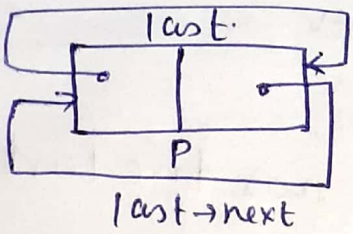
concept →

case 1:- List is empty




```
if (last == NULL)
  printf("Underflow")
```

case 2:- List contains only one node

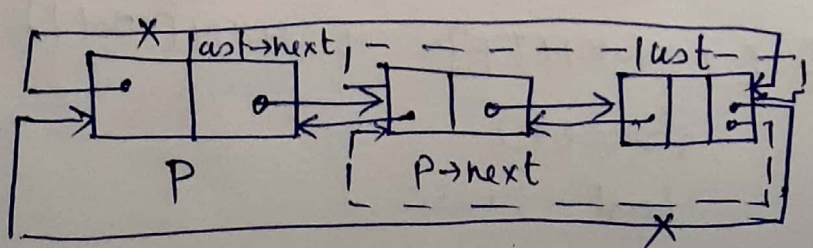



```
p = last→next
if (p→next == last→next)
{
  free(p);
  last = NULL;
}
```

case 3- List contains more than one node

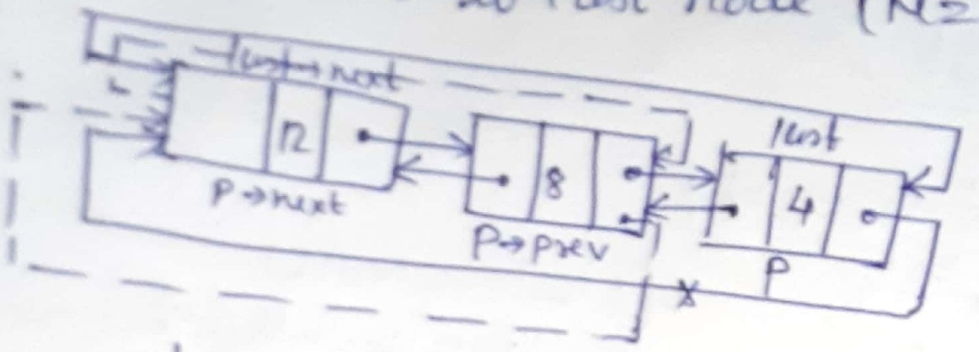
```
p = last→next;
printf("Enter Node number to be deleted");
scanf("%d", &N);
for(i=1; i<N; i++)
    p = p→next.
```

i) p points to first node N=1



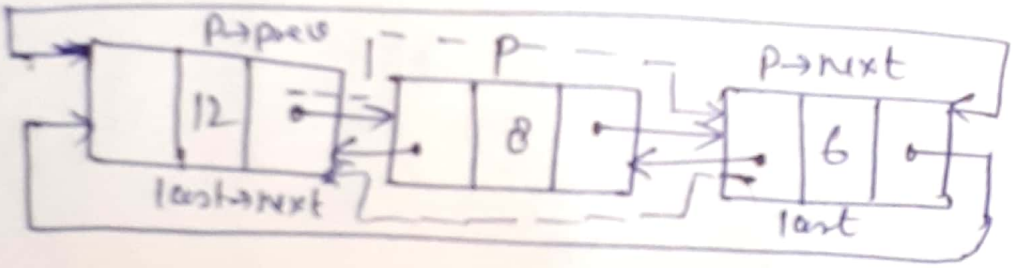

```
if(p == last→next)
{
p→next→prev = last
last→next = p→next
```

ii) P points to last node (N=3)




```
if (P == last)
{
   P→prev→next = p→next
   P→next→prev = p→prev
   last = P→prev
   free(P)
}
```

iii) P points neither to first nor the last (N=2)




```
P→prev→next = P→next
P→Next→prev = P→prev
```

Algorithm→

```
delete_particular_node( )
if (last == NULL)
   write "Underflow"
else exit
   P = LINKN[last]
if ( LINKN[P] == LINKN[last])
      free(P)
      last = NULL
[END of if]
```

else.

p = LINKN[last]

Input N.

for (i=1 to N-1)

p = LINKN[P]

[END of loop]

if (P== LINKN[last])

LINKP [LINKN[P]] = last.

LINKN[last] = LINKN[P]

free(P)

[END of if)

else if (P==last)

LINKN[LINKP[P]] = LINKN[P]

LINKP [LINKN[P]] = LINKP[P]

last = LINKP[P]

free(P)

[END of else if]

else.

LINKN[LINKP[P]] = LINKN[P]

LINKP [LINKN[P]] = LINKP[P]

[END of else]

Exit.

function →

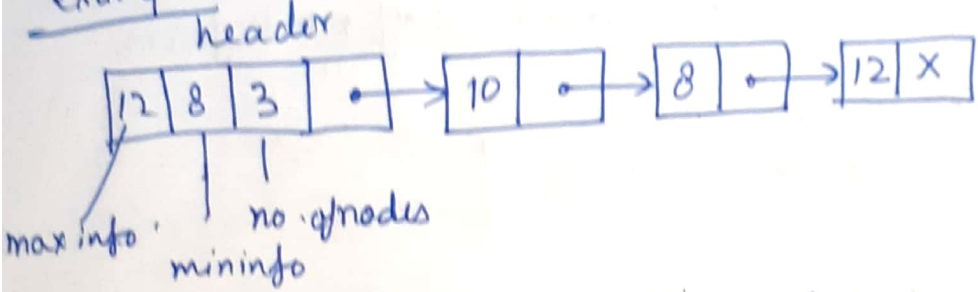
```
Void delete_particular_node()
{
  struct node *p;

  if (last == NULL)
  {
     printf("Underflow");
     exit(0);
  }

  p = last->next;
  if (p->next == last->next)
  {
     free(p);
     last = NULL;
  }
else if (p == last)
  {
     p->prev->next = p->next;
     p->next->prev = p->prev;
     last = p->prev;
     free(p);
  }
else
  {
     p->prev->next = p->next;
     p->next->prev = p->prev;
  }
}
```

Header Linked List →

A header linked list is a variant of Linked List. In a header linked list, a special node present at the begining of a linked list, is called header. This node may contain some information like

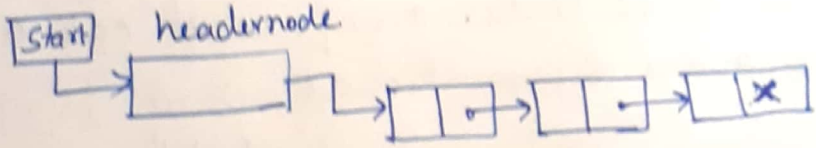
i) Number of nodes in a list
ii) minimum information
iii) maximum information in List
iv) list is sorted or not etc.

example →

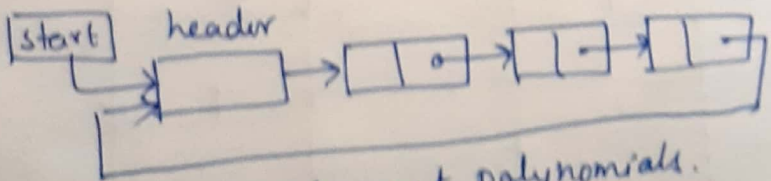


Header Linked List is of two types

1) Grounded Header Linked List - It is a List whose last node contains the NULL pointer.



2) Circular Header Linked List - A List in which last node points back to header node is called circular header linked list.



Applications → to represent polynomials.

Polynomial representation

Linked list is used to represent polynomial expression. In linked list representation of polynomial each node contains three fields.

i) Coefficient field:- holds the value of coefficient of a term.

ii) exponent field :- contains the exponent value of that term.

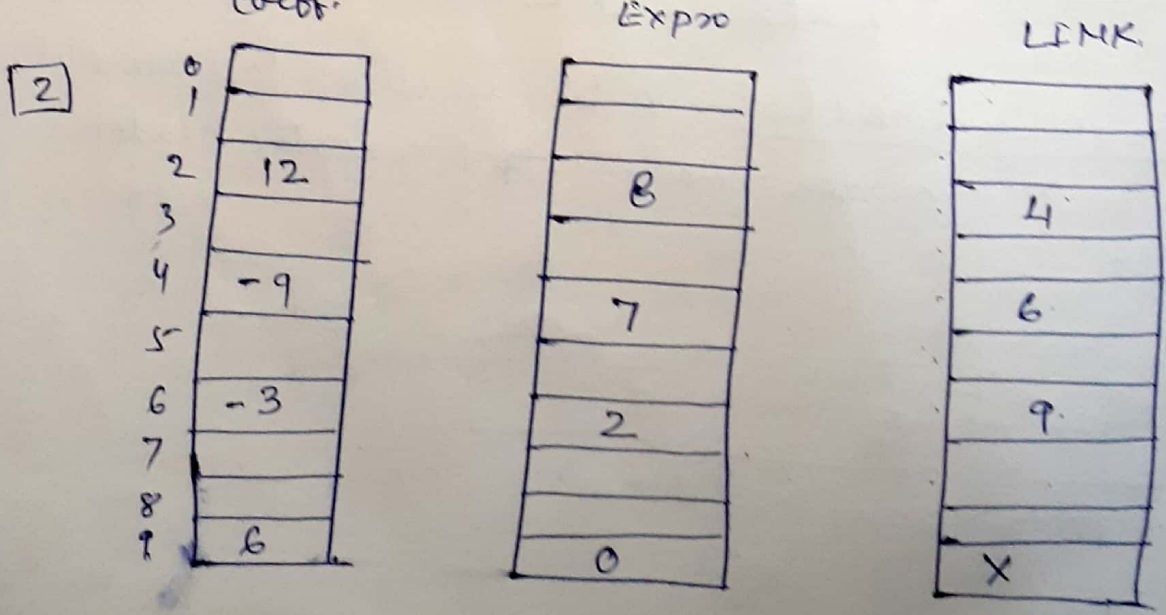
iii) Link field - contains the address of next form.

Q:→ Represent the polynomial

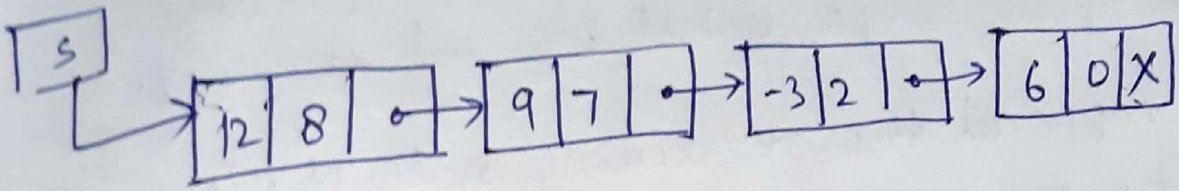
$$P(x) = 12x^8 - 9x^7 - 3x^2 + 6$$

i) using Array ii) using Linked List

Sol: i) It will require three linear arrays which call coeff, expo and LINK and is represented as



ii)



Addition of two polynomials

The steps involved in addition of two polynomials are given as follows:-

i) Read the number of terms in first polynomials. (P)
ii) Read the ~~numbe~~ coefficients and exponents of the first polynomials.
iii) Read the number of terms in second polynomial. (Q)
iv) Read the coeff and exponents of second polynomial.
v) Set the temporary pointers p and q in start two traverse two polynomials P and Q respectively.
vi) Compare the exponents of two polynomials pointed by p and q.

a) if bothe the exponents are equal then add the coefficient and store it in the resultant linked list. and move p and q to next node.
b) if the exponent of current node of P is less than the exponent of current node in q the the coeff. and expo of Second Polynomial is added to the resultant Unked List. and move pointer q to next node.
c) if the exponent of current node of P is greater than the exponent of current node of q, then coeff.

and exponent

∧ is added to resultant linked list and move p to next node.

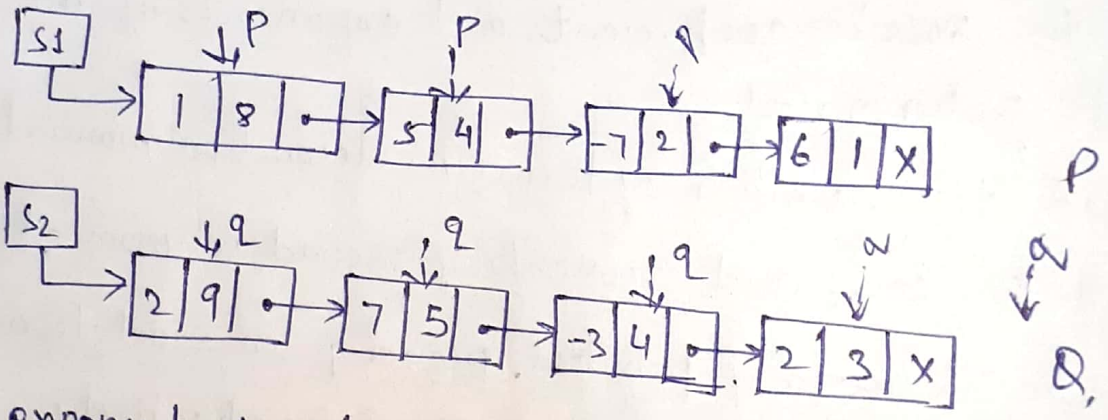
7) Repeat steps unbill one list is empty.

8. Append the remaining nodes of either of the polynomials to the resultant linked list.

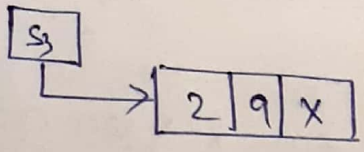
Example→

$$P() = x^8+5x^4-7x^2+6x$$

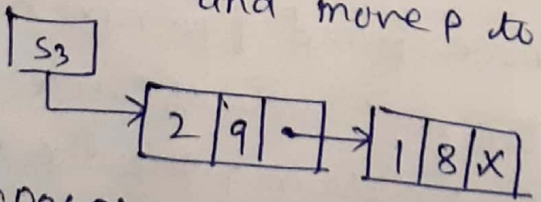
$$Q = 2x^9+7x^5-3x^4+2x^3$$



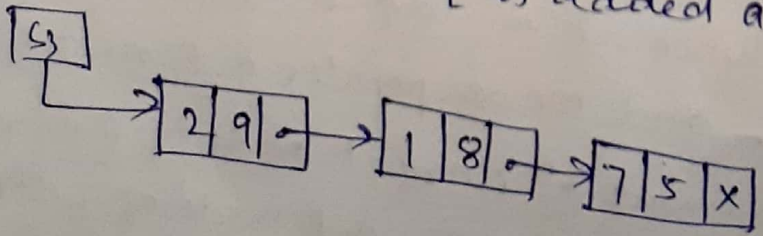
exponent of q is greater than p. so q is added to resultant linked list. and move q to next node



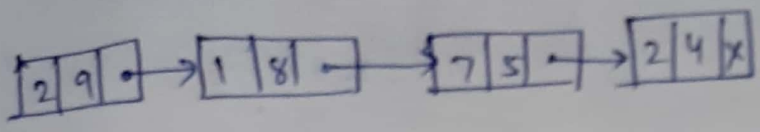
~~expone~~ expo(P) > expo(q), P is added to resultant and move p to next node



expo(q) > expo(P). q is added and move to next node.

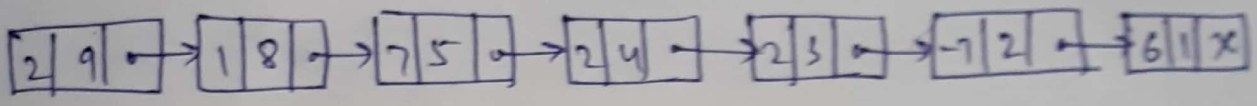


expo(p) = expo(q) so coeff. of both are added and store in resultant list and move p and q to next



expo(q) > expo(p) q is added and move to next.

q is null. so it ends and remaining nodes of p are added to resultant list.



Generalised Linked List →

A generalised linked list is a sequence of $n \geq 0$ elements

i) $A = (a_0, a_1, a_2, \ldots a_{n-1})$ where a_i is either atomic value or a list.

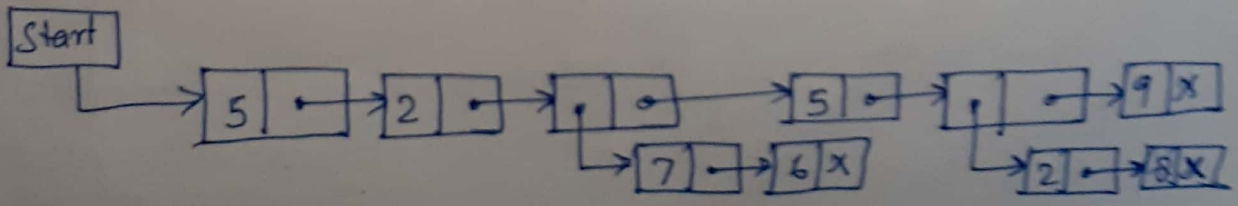
When a_i is list, it is called sublist.

A list may be denoted by a paranthesized enumeration of its elements separated by comma.

example →

1. A = ( ) represents null or empty list having 0 length
2. A = (2, (3,4), 8) list having length 3, there are two atomic value and one sublist having two elements.

Q: make a generalized linked list for (5, 2, (7,6), 5, (2,8), 9)

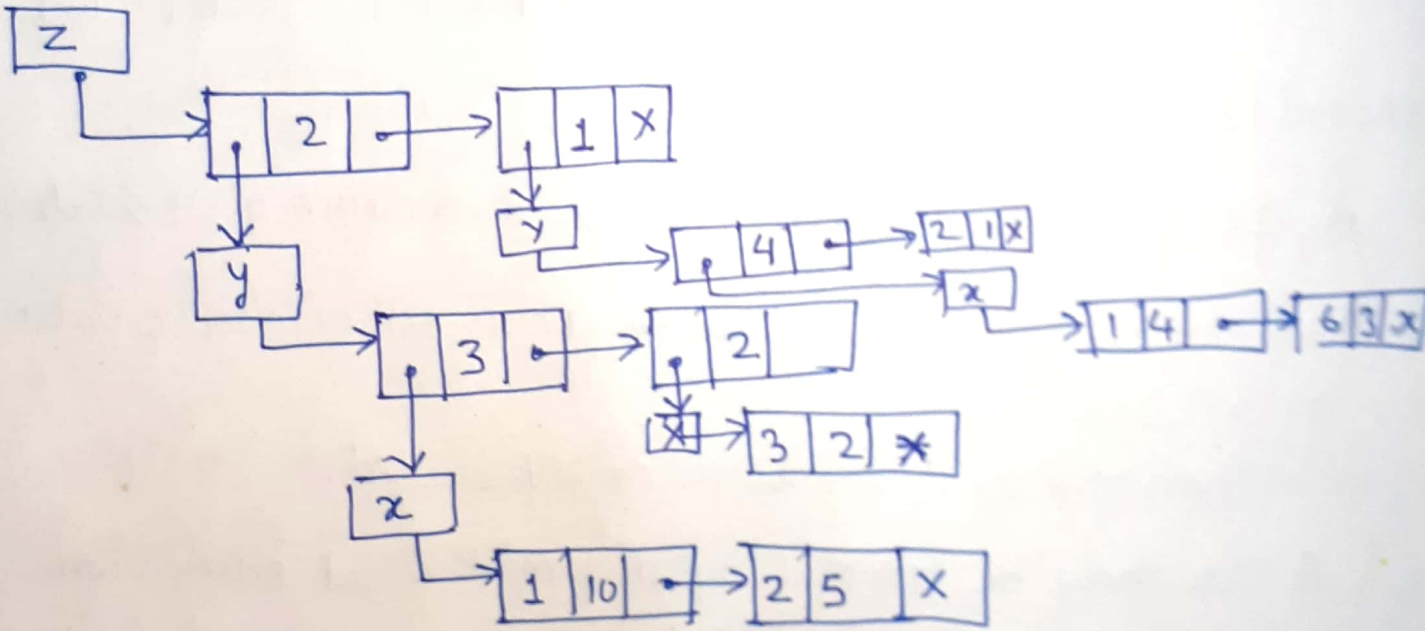


Q:- Consider the polynomial (26)

$$P(x,y,z) = x^{10}y^3z^2 + 2x^5y^3z^2 + 3x^3y^2z^2 + x^4y^4z + 6x^3y^4z + 2yz$$

$$= (x^{10}y^3 + 2x^5y^3 + 3x^2y^2)z^2 + (x^4y^4 + 6x^3y^4 + 2y)z$$

$$= ((x^{10} + 2x^5)y^3 + 3x^2y^2)z^2 + ((x^4 + 6x^3)y^4 + 2y)z$$



Question asked from previous 8 years (AKTU)

Q1:- Define the terms Data structure. List some linear and non-linear data structures stating the application area where they will be used. (2017-18) (2 marks) (2014-15)

Q2:- What do you understand by time space tradeoff? Explain best, worst and average case analysis with example. (2017-18) (7 marks) (2020-21) (2015-16)

Q3:- What are the various asymptotic notations? Explain Big O notation. (2017-18) (7 marks) (2015-16)

Q4:- Write an algorithm to insert a node at the end in a circular linked list. (2017-18) (7 marks)

Q5:- Differentiate Array and Linked list. (2020-21)

Q6:- Consider two dimensional lower triangular matrix (LTM) of order N, obtain the formula for address calculation in the address of row major and column major order of location LTM[i][k] is base address is BA and space occupied by each element is w bytes. (2020-21)

Q7:- Write a c program to insert a node at k^{th} position in a single linked list. (2020-21)

Q8:- How can you represent a sparse matrix in memory? (2019-20)

Q9:- List the various operations on linked list. (2019-20)

Q10:- What are the merits and demerits of array? Given two array of integers in ascending order, develop an algorithm to merge these array to form a third array stored in ascending order. (2019-20) (2016-17)

Q11:- What is doubly linked list? what are its applications (3e) Explain how an elements can be deleted from doubly linked list using C program?
(2019-20)

Q12:- Define the following terms in brief
i) Time complexity iii) Space complexity
ii) Asymptotic notation iv) Big O notation
(2019-20) (2016-17)

Q13:- Differentiate between overflow and underflow condition in a Linked List. (2018-19)

Q14:- What do you mean by various asymptotic notation? Define time space tradeoff. Derive the Big O notation for Linear search. (2018-19)

Q15:- Write a program in C to delete a specific element in a Linked List. Double Linked List takes more space than singly Linked List for storing one extra ~~state~~ address. Under what condition, could a doubly Linked list is more beneficial than singly Linked List. (2018-19)

Q16:- Suppose multidimensional arrays P and Q are declared as P(-2:2, 2:22) and Q(1:8, -5:5, -10:5) stored in column major order.
i) Find the length of each dimension of P and Q.
ii) The number of elements of P and Q.
iii) Assuming base address (Q)=400, w=4. Find the

effective indices E_1, E_2, E_3 and address of element Q[3,3,3]. (2018-19) in row major.

Q17) Differentiate linear and non-linear data structure. (2016-17)

Q18) Give the worst case and best case time complexity of binary search. (2016-17) (2014-15)

Q19) What is meant by circular linked list? Write the function to perform the following operations in a doubly linked list

a) creation of list of nodes
b) Insertion after a specified node
c) Delete the node at given position
d) Sort list according to descending order
e) Display list from the begining to end. (2016-17) (2015-16)

Q20) Given a 2D array A[-100:100, -5:50]. Find the address of A[99,49] considering the base address 10 and each element requires 4 bytes for storage. Follow row major order. (2015-16)

Q21) Explain the application of sparse matrices. (2015-16)

Q22) Consider the linear arrays AAA[5:50], BBB[-5:10] and CCC[1:8]

a) Find the number of elements in each array.

b) Suppose base(A) = 300 and w=4 words per memory cell for AAA. Find the address of AAA[15], AAA[35], and AAA[55]. (2015-16)

Q23) ~~Write overflow condition for circular queue?~~

Q24) Obtain addressing formula for an element in 3D array represented in column major order. (2013-14)

Q25:- Discuss the representation of polynomial of single variable using linked list. Write 'C' function to add two such polynomials re presented by Linked List. (2013-14) (32)

Q26:- A mxn matrix is said to have a saddle point if some entry a[i][j] is the smallest value in row i and the largest value in column j. Write a program that determine the siddle point if one exists. (2013-14)

Q27:- Write a 'C' function that creates a new linear linked list by selecting alternate elements of a given linear linked list. (2013-14)

Q28:- Write a program to reverse a Linked list. (2017-18)

Solution

1) Sell slt-1 (Page 1-2)
2) See slt-1 (page worst, best & average page-5,6, Timespace-15)
3) see set-1 (Page 7-14)
4) See set-4 (Page 18)
5) See-set-2 (Page ~~queue~~ 18)
6) see-set-2 (Page ~~3, back, stack~~ back · page 2 · page 4-5
7) see-set-3 (Page 12)
8) see-set-2 (page 12-15)
9) various operations on Linked List are
   i) Create() → create a list with N nodes.

(ii) Traverse → To visit
exactly once. from start to last node.

(iii) Insertion() → Insert a new node in the list. It is done in 3 ways
a) Insert at begining.
b) Insert at end
c) Insert after a particular node.

(iv) Deletion() → Delete a node from the list. It is also done by 3 ways.
a) Delete first node b) Delete last node.
c) Delete a particular node.

v) Searching → find a particular node with its information

vi) Sorting() → To sort the list either in ascending or in descending order.

vii) merging → To add two lists.

(10) Advantages of array →
i) Elements are accessed randomly. so to access the elements less time is consumed.
ii) searching in array is easy.

Disadvantages →
(i) Array size is fixed so chances of memory wastage.
(ii) Array is homogeneous. so hetrogeneous elements are not stored in an array.

program to merge two sorted array

Q11:- Set - 3 page 11 and set - 4 page 2

Ans12:- Set - 1.

Ans13 - Overflow condition

```
new = (struct node *) malloc (sizeof (struct node));
if (new == NULL)
    printf("overflow");
```

Underflow →

```
if (start == NULL)
    printf("Underflow");
```

Set-6 Pradeep

Ans14- See set-1

Ans15- set-3 page 6

Doubly Linked List is more beneficial when we traverse a list or search a particular element in a list. Because in doubly linked list traversing is done by both ends.

Ans16→ (i) for P

$L_1 = 2-(-2)+1 = 5$ $L_2 = 22-2+1 = 21$

for Q.

$L_1 = 8-1+1 = 8$ $L_2 = 5-(-5)+1 = 11$ $L_3 = 5-(-10)+1 = 16$

(ii) Number of elements in $P = L_1 \times L_2 = 5 \times 21 = 105$

number of elements in $Q = L_1 \times L_2 \times L_3$

$= 8 \times 11 \times 16 = 1408$

(iii) $E_1 = i - LB_i = 3-1 = 2$ $L_1 = 8$

$E_2 = j - LB_j = 3-(-5) = 8$ $L_2 = 11$

$E_3 = k - LB_k = 3-(-10) = 13$ $L_3 = 16$

$Loc(Q[3][3][3]) = base(Q) + w*(E_1L_2L_3 + E_2L_3 + E_3)$

$= 400 + 4*(2 \times 8 \times 11 + 8 \times 11 + 13)$

$= 400 + 4*(176+88+13)$

$= 400 + 4 \times 277$

$= 400 + 1108$

$= 1508$

Ans 17 →

	Linear Data structure	Non Linear
1.	data elements are arranged in a linear order where each and every element attached to its previous and next adjacent except first and last.	data elements are attached in hierarchically manner
2.	single level is involved	multiple levels are involved.
3.	data elements can be traversed in a single run	can't be traversed in a single run.
4.	memory is not utilized in an efficient way	memory is utilized in efficient way
5.	ex→ array, stack, queue	ex→ tree, graph

Ans 18 – worst case complexity – $O(\log N)$

best case → $O(1)$

Ans-19 Set-3 page-12→24.

Ans-20

$L_1 = 100 - (-100) + 1 = 201$

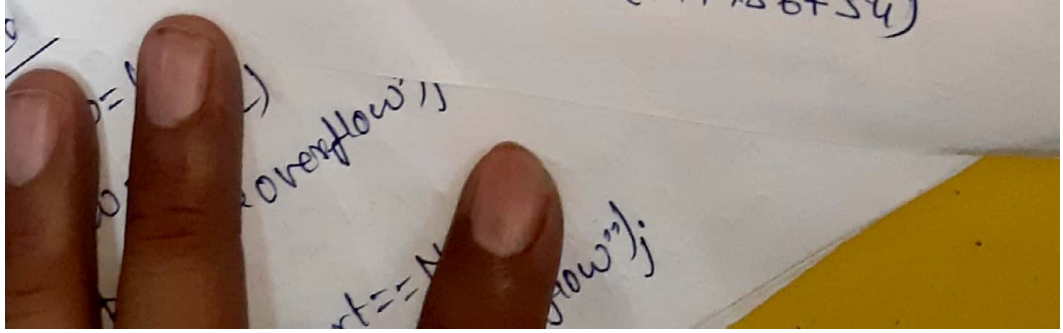
$L_2 = 50 - (-5) + 1 = 56$

$E_1 = 99 - (-100) = 199$

$E_2 = 49 - (-5) = 54$

$LOC[A[99, 49]) = base + w * (E_1 L_2 + E_2)$

$= 10 + 4 * (199 \times 56 + 54)$



Ans21 → Sparse matrices can be useful for computing large scale applications that dense matrices can't handle. for example solving partial differential equation by using the finite element method.

Ans22 → ~~Set-1 page 19~~ Set-1 Page 19

Ans23 - unit-2 Question

Ans24 → Set-2 page 9 and 10

Ans25 → Set-5 → Page 24, 25, 26

Ans26 →

```
main()
{
int i, j, k, m, n, min, max, pos[2][2], A[6][6];
printf("Enter order of matrix");
scanf("%d", &n);
printf("Input matrix");
for(i=0; i<n; i++)
  for(j=0; j<n, j++)
    scanf("%d", &A[i][j]);
for(i=0; i<n, i++)
  {
    min = A[i][0];
    for(j=0; j<m; j++)
    {
      if(min >= A[i][j])
      {
        min = A[i][j];
        pos[0][0] = i;
        p[0][1] = j;
      }
    }
```

```
j = pos[0][1];
max = A[0][j];
for(k=0; k<m; k++)
{
  if(max <= A[k][j])
  {
    max = A[i][j]
    pos[1][0] = k
    pos[1][1] = j;
  }
}
if(min == max)
{
  if(pos[0][0] == pos[1][0] && pos[0][1] == pos[1][1])
    printf("%d", max);
}
}
```

Ans 27 →

```
void create_another()
{
  struct node *p, *new, *current;
  p=start;
new=(struct node *)malloc(sizeof(struct node));
new->next=.
  while(p != NULL)
  {
new=(struct node *)malloc(sizeof(struct node));
  { if(start==NULL)
  new->info = p->info;
  new->next = NULL;
  p=p->next->next;
```

```
current=new;
p=p->next->next;
}
else
{
```

Ans 28→

```
void reverselist()
{
    struct node *p=NULL, *c=start; *n;
    while(c!=NULL)
    {
        n=c->next;
        c->next=r;
        r=c;
        c=n;
    }
}
```