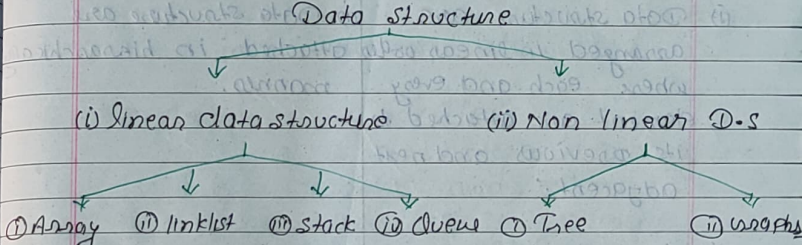
DATA STRUCTURE

PAGE NO.: 1
DATE: 15/03/21

Data structure is a particular way of organizing data in a computer so that we can use effectively.

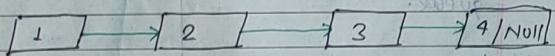
eg:-
we can store a list of items having the same data type using the array data structure.

⇒ Types of data structure:-



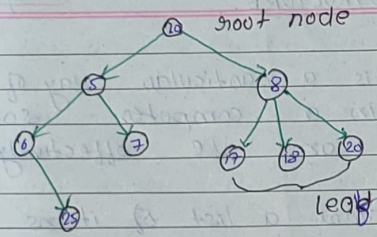
⇒ Difference b/w linear or nonlinear D.S:-

→ Linear ds:- A linear data structure can grow in a sequential organize form.
Examples of linear data structure - linked list, stack, Queue.



link list

→ Nonlinear ds:- In a nonlinear data structure there is no sequential organized way of group.



lead node :- A node without any childs.

Linear ds	Non linear ds
(i) Data structure are arranged in linear order where each and every element are attached to its previous and next adjacent.	(i) Data structure are attached in hierarchically manner.
(ii) Single level is involved.	(ii) Multiple levels are involved
(iii) Memory is not utilized in an effective way.	(iii) Memory is utilized in effective way.
(iv) examples are Array, stack, linklist and queue.	(iv) Examples are Tree, graphs.
(v) Its Implementation is easy.	(v) It's implimentation is complex.
(vi) Application of linear	(vi) Application of non-linear

data structure are mainly used in application of software development.	data structure are AI (Artificial Intelligence) and image processing.

1) Array :- Array is linear data structure that is a collection of similar data.

Types of Array :-

(i) 1D Array :- int a[10];
'a' is an integer array of size 10 so it can store 10 integer from a[0] to a[9].

(ii) 2-D Array :- int a[3][3];
'a' is an integer 2-D array (matrix) of 3 rows and 3 cols so it can store 3*3 = 9 elements from a[0][0] to a[2][2].

⇒ Imp programe inside Array D.S.

(1) WAP in C to store 'n' integers in 1-D array and perform :-
(a) Search
(b) sort
(c) delete an element
(d) display array in reverse order.
(e) sum and avg of array element.
(f) find max and 2nd max array element.
(g) ↑

2.) Add and Multiply two 2-D matrix (2-D array)

3.) Transpose of a Matrix

4.) Right and left Diagonal of a matrix?

5.) Row and column major.

6.) Dynamic Array

7.) Primitive and non primitive Data structure.

8.) Static / Dynamic Data structure

9.) Sparse Matrix

Sparse Matrix

A matrix can be defined as two-dimensional array having 'm' columns and 'n' rows representing m*n matrix. sparse matrices are those matrices that have the majority of their elements equal to zero. In other words, the sparse matrix can be defined as the matrix that has a greater number of zero elements than the non-zero elements.

ex:- 0 0 0
0 4 0
0 0 2

Advantage:- ① less memory required:-

- storage :- As we know, a sparse matrix that contains lesser non-zero elements than zero so less memory can be used to store elements. It evaluate only the non-zero elements.

② computing time will be reduced.

- Computing time:- In the case of searching in sparse matrix, we need to traverse only the non-zero elements rather than traversing all the sparse matrix elements. It saves computing time by logically ~~computing time by logically~~ designing a data structure traversing non-zero elements.

⇒ sparse Matrix Representation:-

- Array Representation.
- linked list Representation.

⇒ Array Representation:-

	0	1	2
0	0	0	13
1	2	0	0
2	3	0	0

Row	col	value
0	2	13
1	0	2
2	0	3

Q.) W.A.P in C to represent sparse matrices using Array.

```
main()
{
    int i, j, r, c, z=0, nz=0, m=0, n=0;

printf("\nEnter row and column");
scanf("%d%d", &r, &c);

int a[r][c];
  for(i=0; i<r; i++)
  {
     for(j=0; j<c; j++)
     {
        scanf("%d", &a[i][j]);
     }
  }
printf("\n Matrix is:\n");

for(i=0; i<r; i++)
{
   for(j=0; j<c; j++)
   {
      if(a[i][j]==0)
      z++;
   else
      nz++;
printf("%d\t", a[i][j]);
}
printf("\n");
```



```
int b[nz][3];
  if(z>nz)
{
   for(i=0; i<r; i++)
{
    for(j=0; j<c; j++)
{
    if(a[i][j] != 0)
{
    b[m][n] = i;
printf("%d\t", b[m][n]);

    b[m][++n] = j;

printf("%d\t", b[m][n]);
    b[m][++n] = a[i][j];

printf("%d\t", b[m][n]);
printf("\n");
m++;
n=0;
}
}
}
else
printf("\n Since zero values are not more
        than non zero values so sparse
        Matrix not possible");
}
```

** Stack :- It is linear data structure that works on LIFO manner.

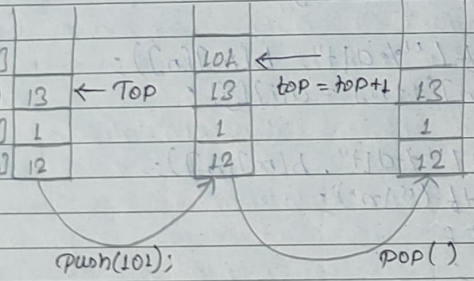
LIFO :- Last In First Out.

→ Operation on stack :-

a) push :- Insert a new element from the end.

b) pop :- delete from the end.

c) peep :- search or traversal.



Note :- In stack insertion and deletion both from top.

→ we can implment stack by two ways :-

a) Array b) link list.

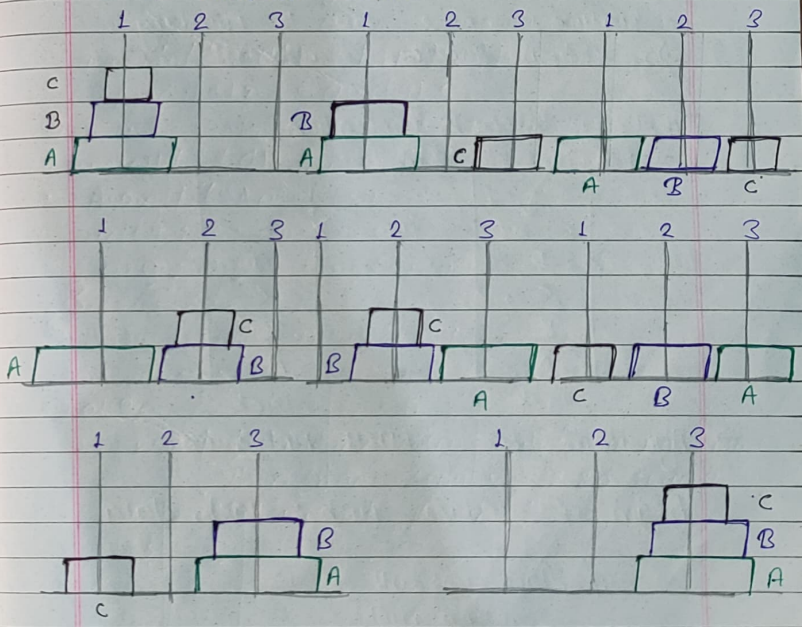
→ stack Application :-

a) Towes of Hanoi

b) Infix, Postfix and postfix conversion.

⇒ Towes of Hanoi :- Towes of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

a) Only one disk can be moved at a time.
b) Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack.
c) No disk may be placed on top of a smaller disk.



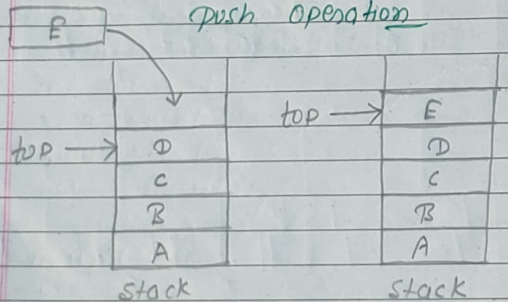
Imp for Col exams

⇒ Push operation:

Add the element from the top in a stack is known as push operation.

→ Push operation involves a series of steps:-

1.) checks if the stack is full.
2.) if the stack is full, produce an error and exit.
3.) If the stack is not full, increments top to point empty space.
4.) Adds data element to the stack location, where top is pointing.
5.) Return success.



→ Algorithm for PUSH Operation:-

```
begin procedure push : stack, data

    if stack is full
        return null
    endif
```



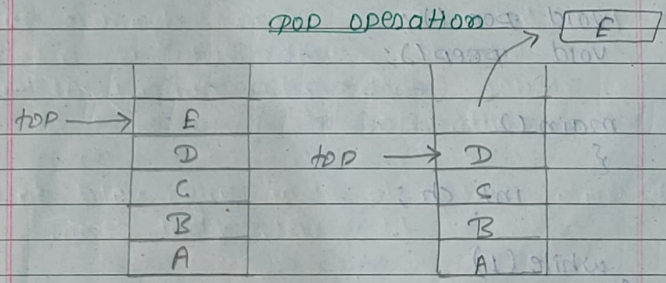
```
    top ← top + 1
    stack [top] ← data

end procedure.
```

⇒ POP Operation:-

Remove the element from the top in a stack is known as pop operation.

→ A pop operation may involves the following steps:-

1) Check if the stack is empty
2) if the stack is empty, produces an error and exit.
3) if the stack is not empty accesses the data element at which top is pointing.
4) Decreases the value of top by 1.
5) Return success.



→ Algorithm for Pop operation:-

```
begin procedure pop: stack

    if stack is empty
        return null
    endif

data ⇐ stack[top]
top ← top - 1
return data

end procedure
```

Q) wap in c to Implement stack?

```
#define max 50
    int stack[max];
int top = -1;
void push();
void pop();
void peep();

main()
{
    int ch;

while(1)
{
printf("\n Push:1 \n Pop: 2 \n Display:3
        \n Exit: 4");
scanf("%d", &ch);
```



```
switch(ch)
{
    case 1:
    push();
    break;

    case 2:
    pop();
    break;

    case 3:
    peep();
    break;

    case 4:
    Exit(0);
  }
}

void push()
 {
    int data;
    if(top == max-1)
        Printf("\n stack is overflow");
  else
  {
    top = top+1;
    Printf("\n Enter data");
    scanf("%d", &data);
    stack[top] = data;
  }
}
```

```
void pop()
{
    if(top == -1)
    printf("\n Stack is under flow");
    else
    {
        printf("\n stack pop data = %d",
                stack[top]);
        top = top - 1;
    }
}
void peep()
{
    int i;
printf("\n Stack Elements \n");
    for(i=top; i>=0; i--)
    printf("%d\n", stack[i]);
}
```

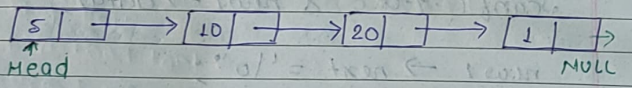
** link list :- It is linear dyanamic data structure, link list is a collection of Nodes, each node consist at least two parts.

a.) Data part :- To store any valid data like:- int, float, char.

b.) Address part :- pointer to store the address of next node so the last node address part filled with NULL.

→ Types of link list :-
- a.) single link list
- b.) circular link list
- c.) double link list
- d.) double circular link list.

A.) single link list :-



Q.) wap in c to create and display single link list?

```
struct node
{ int data;
  struct node *next;
};
typedef struct node s;

main()
{
    s *start, *new, *t;
    int n, i;

printf("\n Enter no. of nodes u want to
        create");
scanf("%d", &n);

    start = (s*) malloc(sizeof(s));
printf("\n Enter data");
scanf("%d", &start->data);
```

```
stant -> next = '\0';
    t = stant;
for(i=1; i<n; i++)
{
    new1 = (s*) malloc(sizeof(s));
printf("\n Enter data");
scanf("%d", &new1 -> data);

    new1 -> next = '\0';
    t -> next = new1;
    t = new1;
}

printf("\n link list elements: \n");
    for(t=stant; t; t=t->next)
        printf("%d", t -> data);
}
```

8) wap in c to insert element in single link list?

```
struct node
{
    int data;
    struct node *next;
};
typedef struct node s;

s *stant, *new1, *t;

void insert()
```



```
void display()
main()
{
    int ch;
while(1)
{
    printf("\n Insert: 1 \n Display: 2 \n
            Exit: 3 \n");
    scanf("%d", &ch)

switch(ch)
{
    case 1:
        insert();
        break;
    case 2:
        display();
        break;
    case 3:
        exit(19191); // meaning will not
            change it still exit(0), no. doesn
            matter.
}
}
}
void insert()
{
    if (stant == '\0')
    {
    stant = (s*) malloc(sizeof(s));
    printf("\n Enter data");
    scanf("%d", &stant -> data);
    stant -> next = '\0';
    t = stant;
```

exit() ✗ wrong
exit(0) ✓ right
should an anys

display() function

जब भी t का next का access करना है तो use
t->next!='\0' in condition. bcoz if you use t!=0
then t का value update होने पर 0 हो जाएगा और t=0 होगा तो t->next possible होगा नहीं

```
    else
    {
      new1 = (s*) malloc(sizeof(s));
      printf("\n Enter data");
      scanf("%d", &new1->data);
      new1->next = '\0';
      t->next = new1;
      t = new1;
    }
}
void display()
{
printf("\n link list elements:\n");
  for(t=start; t->next!='\0'; t=t->next)

  printf("%d:%u->", t->data, t->next);
  printf("%d:%u:", t->data, t->next);
}
```

बड़ा t last में जाएगा तो 0 नहीं होगा t का value last node का point करेगा और t->next possible नहीं होगा

Imp Que:-

1.) sort the link list element using selection sort?
2.) search an element inside link list using binary search?
3.) split the link list into mid point and create two separate link list?
4.) display the prime no. inside the link list?

→ Insert, Display, Primes, sort, search.

```
struct node
{
   int data;
struct node *next;
};
  typedef struct node s;

s* start, *new1, *t;

void insert();
void display();
void sort();
void search();
void Prims();
main()
{
   int ch;
 while(1)
{
printf("\n Insert: 1 \n Display: 2 \n
      Exit: 3 \n primes: 4 \n sort: 5
       \n search: 6 \n");
scanf("%d", &ch);
switch(ch)
{
   case 1: insert();
       break;
   case 2:
       display();
       break;
```

```
case 3:
        exit(0);
case 4:
        prime();
        break;
case 5:
        search(); sort();
        break;
case 6:
        search();
        break;
    }
}

void insert()
{
  if (start == '\0')
  {
  start = (s*) malloc (sizeof (s));
printf ("\n Enter data");
scanf ("%d", &start->data);
  start->next = '\0';
  t = start;
}
else
  {
  new1 = (s*) malloc (sizeof (s));
  printf ("\n Enter data");
  scanf ("%d", &new1->data);
  new1->next = '\0';
    t->next = new1;
    t = new1;
```



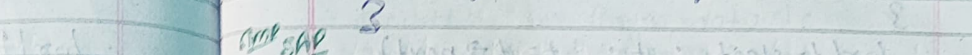
```
void display()
{
 printf ("\n Link list Elements: \n");

 for (t = start ; t->next != '\0' ; t = t->next)

 printf ("%d : %u -> ", t->data , t->next);
 printf ("%d : %u", t->data , t->next);
}
void prime()
{
 int i, f=0;
 printf ("\n Link list only with prime
          data : \n");
 for (t = start ; t; t = t->next)
 {
  for (i=1 ; i<=t->data ; i++)
  {
      if (t->data %i == 0)
        f++;
  }
  if (f == 2)
    printf ("%d : %u -> ", t->data , t->
                                next);
    f=0;
  }
}
```

```
void sort ()
{
    struct node *i, *j;
    int t;
    for(i=start ; i->next != '\0' ; i=i->next)
    {
        for(j=i->next ; j->next != '\0' ;
                                j=j->next)
        {
            if (i->data > j->data)
            {
                t = i->data;
                i->data = j->data;
                j->data = t;
            }
        }
    }

void search()
{
    struct node *i;
    int key, f=0;
    printf("\n Enter data for search");
    scanf("%d", &key);
    for(i=start ; i->next != '\0' ; i=i->next)
    {
        if(i->data == key)
        {
            printf("data found");
            f++;
            break;
        }
    }
    if(f==0)
        printf("data not found");
}
```


Ans:

Q) Represent a polynomial eqn using link list.

eg- $3x^2 + 12x^2 + 4x + 12$




```
struct node
{
    int c, p;
    struct node *next;
};
typedef struct node s;
s *start, *new, *t;
main ()
{
    start = (s*) malloc(sizeof(s));
    printf("\n Enter co-efficient & power");
    scanf("%d%d", &start->c, &start->p);
    start->next = '\0';
    t = start;
    for (int i=1 ; i<=3 ; i++)
    {
        new = (s*) malloc(sizeof(s));
        printf("\n Enter co & po");
        scanf("%d%d", &new->c, &start->p);
```

```
        new1 -> next = '\0';
        t -> next = new1;
        t = new1;
    }
    for(t=start ; t ; t=t->next)
        printf("%d : %d %d->", t->c, t->p);
}
```

SAP Imp

8) Create a link list and split the link list into mid point and creat two separate link list.

```
struct Node
{
    int data;
    struct Node *next;
};
typedef struct Node s;

s *start, *new1, *t, *start2, *start3,
  *t1, *t2;

main()
{
    int n,i;
    printf("\n Enter no. of nodes");
    scanf("%d", &n);
    start1 = (s*) malloc(sizeof(s));
    printf("\n Enter data");
    scanf("%d", &start->data);
    start1 -> next = '\0';
    t = start1;
```



```
    for(i=1 ; i<n ; i++)
    {
        new1 = (s*) malloc(sizeof(s));
        printf("\n Enter data");
        scanf("%d", &new1->data);
        new1 -> next = '\0';
        t -> next = new1;
        t = new1;
    }
    int count = 1;
    for(t = start1 ; t ; t = t->next)
        printf("%d", t->data);

    for(t = start1 ; ; t=t->next)
    {
        if(count == 1)
        {
            start2 = (s*) malloc(sizeof(s));
            start2 -> data = t -> data;
            start2 -> next = '\0';
            t1 = start2;
        }
        else
        {
            new1 = (s*) malloc(sizeof(s));
            new1 -> data = t -> data;
            new1 -> next = '\0';
            t1 -> next = new1;
            t1 = new1;
        }
```

```
if (count >= n/2)
    break;
    count++;
}
    int count1 = 1;

for (t1 = t->next; t1; t1 = t1->next)
{
    if(count1 == 1)
{
    start3 = (s*) malloc (sizeof (s));
    start3 -> data = t1 -> data;
    start3 -> next = '\0';
      t2 = start3;
}
    else
    {
      new1 = (s*) malloc(sizeof (s));
    new1 -> data = t1 -> data;
     new1 -> next = '\0';
      t2 -> next = new1;
       t2 = new1;
    }
     count1++;
}
      printf ("\n");
for(t = start2; t; t = t->next)
    printf ("%d", t->data);
      printf ("\n\n");

for(t = start3; t; t = t->next)
    printf ("%d", t->data);
```

Raw Major and column major order

Q) Addition of two polynomial eqn. using link list

2-D array compiler can flow row major and column major to assign address.

Note:- Array always created in a single dimension in memory because memory is linear and have single integer array.

Raw and column major order:- In computing row and column major order are method for storing multidimensional arrays in linear storage such as random access memory.

⇒ Raw Major of 2-D Array:-

array:-
int a[3][4]

	0	1	2	3
0	100	102	104	106
1	108	110	112	114
2	116	118	120	122

⇒ Raw Major:-

a_{00}	a_{01}	a_{02}	a_{03}	a_{10}	a_{11}	a_{12}	a_{13}	a_{20}	a_{21}	a_{22}	a_{33}
100	102	104	106	108	110	112	114	116	118	120	122

Address int(a[i][j]) = $l_0 + [i*n + j] * w$
{when index starts from 0}

Address of int(a[i][j]) = $l_0 + [(i-1)*n + (j-1)] * w$
{when index starts from 1}

where, l_0 is base address
i is index of row
j is index of column
n = no. of column
w = size of data type.

Q) Find the row major of 2-d array int a[2][2]

formula = $l_0 + [i*n + j] * w$

$= 100 + (2*4 + 1) * 2$
$= 100 + (8+1) * 2$
$= 100 + 18$
$= 118$ Ans

⇒ column major:-

add int(a[i][j]) = $l_0 + (j*m + i) * w$
for index starts from 0.

add int(a[i][j]) = $l_0 + ((j-1)*m + (i-1)) * w$
for index starts from 1.

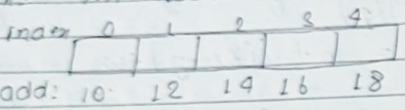
where, m = no. of row
l_0 = base address
i = row index
j = column index
w = size of data type.

8) find the column maggor of add[0][2] where index starts from 0:

= $l_0 + (j * m + i) * w$
= 100 + (2 * 3 + 1) * 2
= 100 + (7 * 2)
= 114

⇒ Finding the row maggor of single dimensional array:-

ex:- int a[5].



add of (a[i]) = base add + i * size of data type
= $l_0 + i * w$ (for array index starts from 0)
= $l_0 + (i-1) * w$ (for array index starts from 1)

where, l_0 = base add
i = array index whose address we want to find.
w = size of data type.

add(a[3]) = $l_0 + i * w$ consider index
= 10 + 3 * 2 starts from 0.
= 16

add(a[4]) = $l_0 + (i-1) * w$ consider array
= 10 + (4-1) * 2 index starts
= 16 from 1.

8) Implement sparse matrix using linked list?

```
main()
{
  int a[3][3] = {{0,0,0}, {0,1,0}, {0,0,5}};
  int i, j, c=0, c1=0, x=0;

for(i=0; i<3; i++)
 {
   for(j=0; j<3; j++)
    {
      if (a[i][j]!=0)
        c++;
      else
        c1++;
    }
 }
 if ( c1 > c)
 {
    struct node
    {
      int r, c, v;
      struct node *next;
    };
 typedef struct node s;

 s *start, *new1, *t;
```

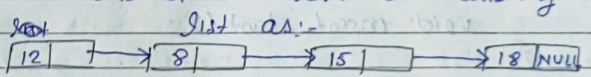
```
for(t=start ; t->next != '\0' ; t=t->next)
{
    printf("%d %d %d ->", t->r, t->c, t->v);
}
printf("%d %d %d ->", t->r, t->c, t->v);
}
else
{
    printf("since non zero values are more than zero values so sparse matrix is not possible");
}
}
```

* Insert a new node in a existing link list:-

1.) Insert node at start:-

⇒ Algo:-

consider we have an existing link list as:-



Step1:-

Create a new node:-

```
new1 = (s*) malloc (sizeof(s));
printf("enter data:");
scanf("%d", &new1->data);
```

Step2:- Make new Node as start node:-

```
new1->next = start;
```

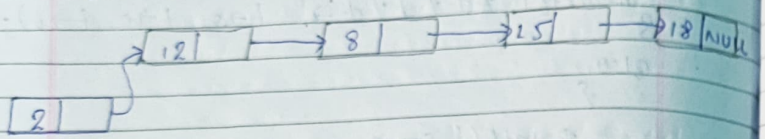


```
for(i=0; i<3; i++)
{
    for(j=0; j<3; j++)
    {
        if(a[i][j] != 0)
        {
            x++;
            if(x==1)
            {
                start = (s*)malloc(sizeof(s));
                start->r = i;
                start->c = j;
                start->v = a[i][j];
                start->next = '\0';
                t = start;
            }
            else
            {
                new1 = (s*)malloc(sizeof(s));
                new1->r = i;
                new1->c = j;
                new1->v = a[i][j];
                new1->next = '\0';
                t->next = new1;
                t = new1;
            }
        }
    }
}
printf("\n sparse matrix : \n");
```

step3:- Make the new node as start:-

start = new1;

Now the link list as:-



where,

```
struct node
{
    int data;
    struct node *next;
};
typedef struct node S;
S *start, *new1;
```

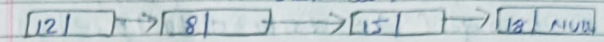
function:-

```
void insert_start()
{
    new1 = (S*) malloc (sizeof (S));
    printf ("\n Enter data");
    scanf ("%d", &new1->data);
    new1->next = '\0';
    new1->next = start;
    start = new1;
}
```


2) Insert node at end:-

→ Algo:-

Consider we have an existing link list as:-



step1:-

Create a new node:-

```
New1 = (S*) malloc (sizeof (S));
printf ("enter data");
scanf ("%d", &new1->data);
new1->next = '\0';
```

~~step2:- Make new node as start node:-~~

```
new1->next = start;
```

~~step3:- Make the new node as start:-~~

&

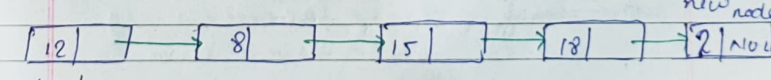
step2:- Move a pointer and terminate at last node so it hold the address of last node:-

```
for (t = start; t->next != '\0'; t = t->next)
{}
```

step 3:- Add the new node as last node:

```
t->next = new;
```

Now, the link list as:-



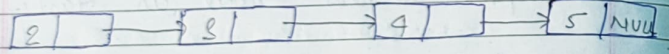
function :-

```
void insert_end()
{
    new1 = (s*) malloc(sizeof(s));
    printf("\n Enter data");
    scanf("%d", &new1->data);
    new1->next = '\0';

for(t=start; t->next!='\0'; t=t->next);
    t->next = new1;
}
```

3.) Insert Node at specified position (sp) :-

⇒ Algo :-

Consider we have an existing link list :-



step1 :- create a New node :-

```
new1 = (s*) malloc(sizeof(s));
printf("enter data");
scanf("%d", &new1->data);
new1->next = '\0';
```

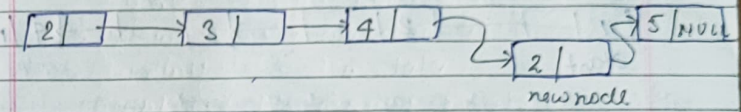
step2 :- printf("\n enter node data after which U want to add new node");

```
scanf("%d", &k);
```


step3 :- Insert new node at specified position :

```
for(t1=start; t2=t1->next; t1->data!=k;
    t1=t1->next, t2=t2->next);
    t->next = new1;
    new1->next = t2;
```

⇒ So if k=4 then link list :-



⇒ function :-

```
void insert_sp()
{
    int k;
    s*t1, *t2;

new1 = (s*) malloc(sizeof(s));
printf("\n enter data");
scanf("%d", &new1->data);
    new1->next = '\0';

printf("\n Enter node data after which
    you want to add new node");
scanf("%d", &k);

for(t1=start, t2=t1->next; t1->data!=k;
    t1=t1->next, t2=t2->next);
    t1->next = new1;
    new1->next = t2;
}
```

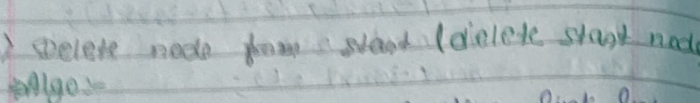
Delete Node

⇒ Delete a node from an existing link list.

a) Delete node from start (delete start node):

Algo:-

consider we have an existing link list as:-



step 1:- store the address of start node in a temp variable:-

```
t = start;
```

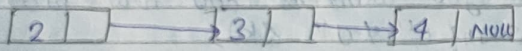
step 2:- Move the start to the 2nd node:-

```
start = start->next;
```

step 3:- Remove the start node:

```
free(t);
```

Now this link list is as follows:-



⇒ function:-

```
void delete_start()
{
    t = start;
```

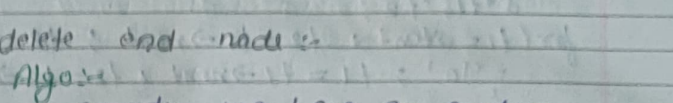


```
    start = start->next;
    free(t);
}
```

b) delete end node:-

⇒ Algo:-

consider we have an existing link list as:-



step 1:- Move two pointers so that when the loop is terminated.

1st pointer at 2nd last node and end pointer at last node:-

```
for(t1=start, t2=t1->next; t2->next!=NULL;
    t1=t1->next, t2=t2->next);
```

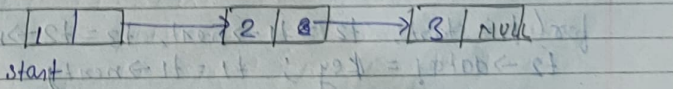
step 2: set the t1 next to null

```
t1->next = NULL;
```

step 3:- delete the last node;

```
free(t2);
```

Now the link list is as:-



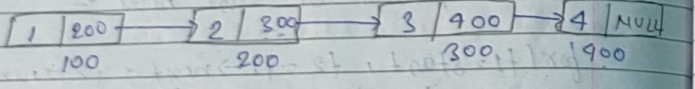
function :-

```
void delete_end()
{
    s*t1, *t2;
for(t1=start ; t2=t1->next ; t2->next!
   ='10' ; t1=t1->next , t2=t2->next);
    t1->next='10';
    free(t2);
}
```

c) Delete the specified node :-

⇒ Algo :-

consider we have an existing link list as :-



Step 1 :- int key;
printf("Enter the node data U want to delete");
Scanf("%d", &k);

Step 2 :- More three pointer so that when the loop is terminated, t1 will at just before the key node, t2 at key node and t3 at just after the key node.

```
for(t1=start, t2=t1->next, t3=t2->next;
  t2->data!=key ; t1=t1->next,
      t2=t2->next, t3=t3->next);
```

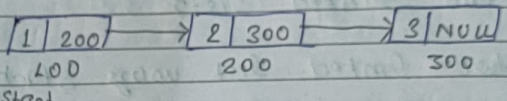

Step 3 :- Assign the address of t2 to t1;

```
t1->next=t3;
```

Step 4 :- delete the key node;

```
free(t2)
```

Step Now the link list (if key=3)



⇒ function :-

```
void delete_sp()
{
    s*t1, *t2, *t3;
    int key;
printf("\n Enter node data U want to
                delete");
scanf("%d", &key);
for(t1=start ; t2=t1->next, t3=t2->next;
  t2->data!=key ; t1=t1->next , t2=t2->next,
     t3=t3->next);
  t1->next=t3;
  free(t2);
}
```

1) Find the middle element of a singly linked list in one pass?

2) Delete the Duplicate data nodes in a linked list?

3) Reverse the linked list?

4) Add two linked lists?

5) Add two linked list using stack?

6) Print Alternate node data?

7) Print even & odd node data?

8) Merge two sorted linked list?

9) Print nth node data entered by user?

2nd method

⇒ function :- delete frompos()
{
struct node * nextnode, *head, *temp;
int pos, i=1;
temp = head;
printf("enter position");
scanf("%d", &pos);

while (i < pos - 1)
{
temp = temp → next;
i++;
}

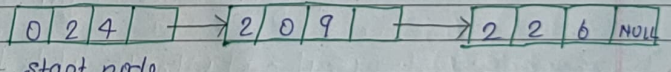
nextnode = temp → next;
temp → next = nextnode → next;
free (nextnode);
}

⇒ The following are the fields in the linked list:-

Row :- It is an index of row where a non-zero element is located.
column :- It is an index of column where a non-zero element is located.
value :- It is the value of the non zero element which is located at the index(row, column).
Next node :- It stores the address of the next node.

ex:-

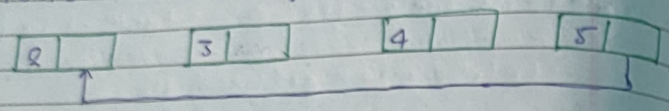
	0	1	2
0	0	0	4
1	0	0	0
2	9	0	6



start node

Circular Linked list

Circular linked list is a variation of linked list in which the last element points to the first element.



Q:) Create and display circular link list ?

```
Struct node
{
   int data;
  Struct node *next;
};
typedef struct node S;
  S *start, *t, *new1;

void insert();
void display();

 main()
{
   int ch;
   while (1)
{
   printf ("\n Insert: 1 \n Display : 2 \n");
    scanf ("%d", &ch);

 Switch (ch)
{
   case 1:
   insert();
   break;
   case 2:
    display();
    break;
}
}

  void insert()
{
    if (start == '\0')
{
   start = (s*)malloc (sizeof (s));
   printf ("\n Enter data");
   scanf ("%d", &start->data);
    start -> next = start;
    t = start;
}
else
{
   new1 = (s*)malloc [sizeof(s)];
   printf ("\n Enter data");
   scanf ("%d", &new1->data);
    t -> next = start; new1;
    t = new1; new1->next = start;
}                    t = new1;
}
```

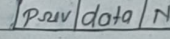
```
void display()
{
printf("\n Circular Link List : \n");
for(t=start; t->next!=start; t=t->next)
{
    printf("%d" : %", t->data);
}
printf("%d : ", t->data);
printf("%d : ", t->next->data);
}
```

Doubly linked list

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts:

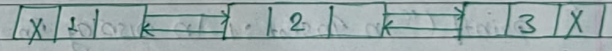
1.) node data
2.) Pointer to the next node (forward pointer)
3.) pointer to the previous node (backward ")

- A Sample node in a doubly linked list is shown in the figure:-



Node

- A doubly linked list containing three nodes having numbers from 1 to 3 in their data part is shown:-



doubly linked list.

⇒ Node of a doubly linked list:-

```
struct Node {
    int data;
    struct Node *fp;
    struct Node *bp;
};
```

8) W.A.P in C to implement (create & display) the doubly linked list?

```
struct Node
{
    int data;
    struct Node *fp, *bp;
};
typedef struct Node S;

void insert();
void display();
void display_rev();

S* start, *new1, *t;
main()
{
  int ch;
while (1)
{
  printf("\n Insert : 1 \n Display \n
        Reverse order : 2 \n Display : 3 \n");

  scanf("%d", &ch);

  switch (ch)
    {
        case 1:
        insert();
        break;
```



```
        case 2:
        display_rev();
        break;

        case 3:
        display();
        break;
    }
  }
}
void Insert()
  {
      if(start == '\0')
    {
      start = (s*)malloc(sizeof(s));
      printf("\n Enter data");
        scanf("%d", &start->data);
      start->fp = '\0';
      start->bp = '\0';
        t = start;
    }
else
    {
      new1 = (s*)malloc(sizeof(s));
      printf("\n Enter data");
      scanf("%d", &new1->data);
        new1->fp = '\0';
        new1->bp = '\0';
        t->fp = new1;
        new1->bp = t;
          t = new1;
    }
}
```

```
void display-rev()
{
printf("\n Reverse order: \n");
for( ; t->bp != '\0'; t = t->bp);
  {
    printf("%d\t", t->data);
  }
    printf("%d", t->data);
}

void display()
  {
    printf("\n Elements : \n");
for( t=start; t->fp != '\0'; t = t->fp)
  {
    printf("%d\t", t->data);
  }
    printf("%d", t->data);
}
```

refer rough

Q.) Add two br poly^{al} eq^{n} using linked list?

Stack Implementation

⇒ WAP in C to perform all the operation of stack using linked list?

a.) push b.) pop c.) peep (Traversal)

```
Struct Node
{ int data;
  struct Node *next;
};
    typedef struct Node s;

s *start, *top, *new1, *t1, *t2;
void push();
void pop();
 void peep();

main()
{
  int ch;
while(1)
{
  printf("\n push : 1 \n pop : 2 \n peep : 3 \n
   scanf("%d", &ch);
switch(ch)
{
  case 1:
   push();
   break;
  case 2:
   pop();
  break;
```

```
        case 3:
        peep();
        break;
    }
}
void push()
{
    if(start == '\0')
    {
    start = (st*)malloc(sizeof(s));
    printf("enter data \n");
    scanf("%d", &start->data);
    start->next = '\0';
      t = start;
    }
    else
    {
    new1 = (st*)malloc(sizeof(s));
    printf("enter data \n");
    scanf("%d", &new1->data);
     new1->next = '\0';
      top->next = new1;
        top = new1;
    }
}
void pop()
{
    if(start == '\0')
  printf("\n stack is under flow");
else
    {
```



```
printf("\n pop data : %d", top->data);
for(t1 = start; t2 = t1->next; t2->next != '\0';
    t1 = t1->next, t2 = t2->next);

    t2->next = '\0';
    free(t2);
    top = t1;
}
}
void peep()
{
printf("\n stack elements are : \n");
for(t1 = start; t2->next != '\0'; t1 = t1->next)
    {
    printf("%d : ", t1->data);
    }
  printf("%d", t1->data);
}
```

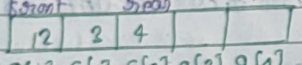
Queue

It is linear data structure that works on FIFO manner.
where FIFO means "First In First Out".

Implementation:
1.) Array imp" (static queue)
2) Link list imp"(Dyanamic queue)

⇒ Types of Queue:-

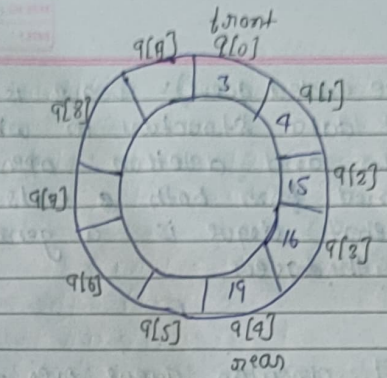
1) Straight queue(linear queue):-



⇒ In linear queue insertion always happen from rear and deletion from front.

2) circular Queue:- In circular queue all the nodes are represented as circular. It is similar to the linear except that the last element of the queue is as ring buffer as all the ends are connected to another ends.

⇒ The circular queue can be represented as:-



3) Priority Queue:- A priority queue is another special type of Queue data structure in which each element has some priority associated with it. Based on the priority of the element, the elements are arranged in a priority queue. If the elements occur with the same priority, then they are served according to the FIFO principle.

⇒ linked list data structure of priority queue:-

```
struct node
{
    int data, pr_no;
    struct node *next;
};
```

linked list Representation

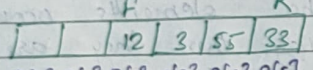
⇒ Array data structure:-

```
struct c_queue
{   int data;
    int p_no;
} c q[10];
```

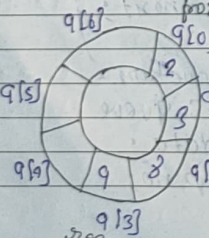
4) Deque (Double ended queue):- Deque is a linear data structure in which the insertion and deletion operation are performed from both ends. we can say that deque is a generalized version of the queue.

5) Advantage of circular queue over the linear queue?

Ans⇒ consider a case in linear queue (Array implemented) where, there are some vacant space available from the front end but full rear at the end i.e rear = max-1.



Since rear is on max-1 so we can not insert any new elements, that creates wastage of memory area. To solve this problem we use the circular queue.



8) Create a linear queue using array and perform:-

a) insert element from rear (Algorithm) (college)

b) delete from front (Algo) college

a)
```
#define max 10
int r = -1;
int f = -1; int q[max];
void insert(int);
void delete();
void display();

main()
{
    int ch, data;
  while(1)
  {
    printf("\n Insert: 1 \n Delete: 2 \n Display: 3 \n");
    scanf("%d", &ch);

switch(ch)
{
    case 1:
      printf("\n Enter data");
      scanf("%d", &data);
      insert(data);
      break;
```

```
        case 2:
            delete1();
            break;
        case 3:
            display();
            break;
        }
    }
}
void insert(int x)
{
    if (r == max-1)
    printf("\n Queue is full");

    else if (r == -1 && f == -1)
    {
        r=0;
        f=0;
        q[r] = x;
    }
    else {
        r++;
        q[r] = x;
    }
}
void delete1()
{
    if (r == -1 && f == -1)
        printf("\n Queue is empty");

    else if (r == 0 && f == 0)
        printf("\n Deleted Data : %d", q[f]);
```



```
        f = -1;
        r = -1;
    }
    else {
        printf("\n Deleted Data : %d", q[f]);
        f = f+1;
    }
}
void display()
{
    int i;
    printf("\n Queue elements are: \n");

    for(i=f; i<=r; i++)
        printf("%d\t", q[i]);
}
```

Algo=

8) wap in c++ to implement queue using linked list?

```
Using namespace std;
#include<iostream>

struct node
{
    int data;
    struct node *next;
};
typedef struct node s;

s *f, *r, *newl, *tl;

void insert(int);
void delete1();
void display();

main()
{
    int ch, d;
while(1)
{
  cout<<"\n Insert:1 \n delete:2 \n
        display:3 \n 4";
    cin>>ch;

switch(ch)
{
    case 1:
    cout<<"\n Enter data";
    cin>>d;
```

new syntax
ptr = new type;
ex:
*r = new s;
delete syntax
delete ptr;
ex:- delete tl;

```
else{
    t1 = f;
    f = f->next;
    delete t1;
  }
}
void display()
{
  cout<<"\n Queue elements\n";
  for(t1=f; t1->next!=NULL; t1=t1->next)
    cout<<t1->data<<"\t";
    cout<<t1->data;
  }
```

9) wap in c++ to implement circular queue using link list ?

```
Using namespace std;
#include<iostream>
 struct node
{
   int data;
 struct node * next;
};
  typedef struct node s;

s *f, *r, *new1, *t1;

void insert(int);
void display();
void deleteL();
```



```
    insert(d);
    break;
  case 2:
    deleteL();
    break;
  case 3:
    display();
    break;
  }
}
void insert(int x)
{
  if(r==NULL && f==NULL)
  {
    f=new s;
    f->data = x;
    f->next = NULL;
    r=f;
  }
  else{
    new1 = new s;
    new1->data = x;
    new1->next = NULL;
    r->next = new1;
    r = new1;
  }
}
void deleteL()
{
  if (r==NULL && f==NULL)
    cout<<"\n Queue is empty";
```

```
#define max 10;
int q[max];
int r = -1;
int f = -1;
void delete1();
void insert();
void display();

main()
{
  int ch;
while(1)
{
  printf("\n Insert:1 \n Delete1:2 \n
          display:3\n");
  scanf("%d", &ch);

switch(ch){
    case1:
    insert();
    break;

    case2:
     delete1()
     break;
    case3:
     display();
     break;
   }
}
}
```



```
void Insert()
{
  int d;
  if(r==max-1)
   printf("\n Queue is full");

 else if(r==-1 && f==-1)
  {
    printf("\n Enter data");
    scanf("%d", &d);
     r=f=0;
     q[r]=d;
  }
 else{
    printf("\n Enter data");
    scanf("%d", &d);
     r=r+1;
     q[r]=d;
  }
}
 void delete1()
  { int i;
   if(r==-1 && f==-1)
     printf("\n Queue is empty so can
             not delete data");
    else{
   printf("\n Deleted data:%d", q[f]);

   for(i=f; i<=r; i++)
    {
      q[i]=q[i+1];
    }
```

```
        r = r - 1;
    }
}
void display()
{
    int i;
printf("\n Queue elements are: \n");
  for(i=f; i<=r; i++)
    {
      printf("%d\t", q[i]);
    }
}
```

V.V.I (exam)

** Infix to postfix Using stack

Operator precedence	Associativity
1. ( )	
2. ^	Right → left } No need to pop out
3. /*	left → Right } pop out
4. +-	left → Right } pop out

Rules for conversion:

1. If the lower priority operator enter in the stack then pop out the higher priority operator from the stack.
2. If two equal priority operator are in stack then check the associativity, if the Asso" if is left to Right then pop out otherwise stay in the stack.
3. If the closing parenthesis come then pop out all the operator between parenthesis.

Q.) Infix to postfix conversion:-

K+L-M*N+(O^P)*W/U/V*T+Q

Infix exp"	stack	postfix exp"
K		K
+	+	K
L	+	KL
-	-	KL+

M	-	KL+M
*	-*	KL+M
N	-*	KL+MN
+	+	KL+MN*-
(	+(	KL+MN*-
O	+(	KL+MN*-O
^	+(^	KL+MN*-O
P	+(^	KL+MN*-OP
)	+$	KL+MN*-OP^
*	++	KL+MN*-OP^
W	++	KL+MN*-OP^W
/	+/	KL+MN*-OP^W*
U	+/	KL+MN*-OP^W*U
/	+/	KL+MN*-OP^W*U/
V	+/	KL+MN*-OP^W*U/V
*	++	KL+MN*-OP^W*U/V/
T	++	KL+MN*-OP^W*U/V/T
+	+	KL+MN*-OP^W*U/V/T*+
Q	+	KL+MN*-OP^W*U/V/T*+Q

= KL+MN*-OP^W*U/V/T*+Q+

Ans

Rule 1:- In the conversion of Infix to postfix the lower priority operator can followed by the higher priority operator in the stack.

Rule 1.1:- In the stack of infix to postfix conversion the higher precedence operator never be followed by lower precedence operators.

8) a+b*(c^d-e)^(f+g*h)-i

Infix Input	stack	Postfix
a		a
+	+	a
b	+	ab
*	+*	ab
(	+*(	ab
c	+*(	abc
^	+*(^	abc
d	+*(^	abcd
-	+*(-	abcd^
e	+*(-	abcd^e
)	+*	abcd^e-
^	+*^	abcd^e-
(	+*^(	abcd^e-
f	+*^(	abcd^e-f
+	+*^(+	abcd^e-f
g	+*^(+	abcd^e-fg
*	+*^(+*	abcd^e-fg
h	+*^(+*	abcd^e-fgh
)	+*^	abcd^e-fgh*+
-	-	abcd^e-fgh*+^*+
i	-	abcd^e-fgh*+^*+i

= abcd^e-fgh*+^*+i-

Ans

8) a+(b*c(d/e^f)*g)*h

Input	Stack	Postfix
a		a
+	+	a
(	+(	a
b	+(	ab
*	+(*	ab
c	+(*	abc
(	+(*(	abc
d	+(*(	abcd
/	+(*(/	abcd
e	+(*(/	abcde
^	+(*(/^	abcde
f	+(*(/^	abcdef
)	+(*	abcdef^/
*	+(*	abcdef^/*
g	+(*	abcdef^/*g
)	+	abcdef^/*g*
*	+*	abcdef^/*g*
h	+*	abcdef^/*g*h
		abcdef^/*g*h*+

** Converting Infix to Prefix

Step1:- Reverse the Infix expression i.e A+B*C will become C*B+A. Note while reversing each '(' will become ')' and each ')' become '('.

Step2:- Obtain the reverse postfix expression of the modified expression i.e CB*A+.

Step3:- Reverse the postfix expression. Hence in our example prefix will +A*BC.

8) Infix
A+B*C postfix = ?

⇒ Reversing C*B+A

⇒ finding postfix of the modified exp:-

Input	Stack	Postfix
C		C
*	*	C
B	*	CB
+	+	CB*
A		CB*A
		CB*A+

⇒ Again reversing for prefix

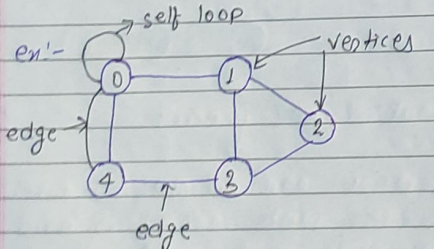
= +A*BC

Graph

A graph is a collection of two sets V and E where V is finite non-empty set of vertices and E is a finite non-empty set of edges.

- Vertices are nothing but the nodes in the graph.
- Two adjacent vertices are joined by edges.
- Any graph is denoted as

$$G = \{V, E\}$$

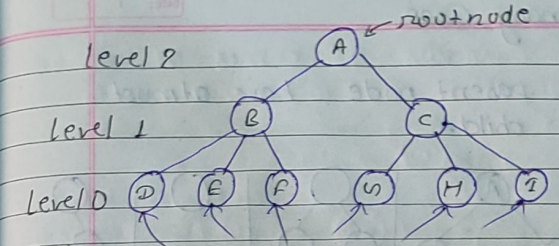


Tree

A tree is a finite set of one or more nodes such that:

1) There is a specially designed node called root.

2) The children of the parent node (root) are known as subtrees.



∴ Height = 2

Leaf nodes : Having no any child Node.

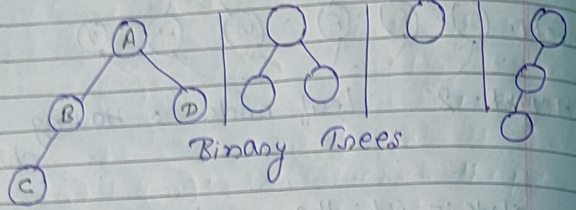
Note:- If the number of nodes is n then no. of edges must be n-1 in the tree.

8) Diff between Graph & Tree

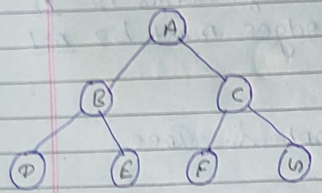
Graph	Tree
1. There is no unique node called root in Graph.	1. There is a unique node called root in trees.
2. A cycle can formed.	2. There will not any cycle.
3. App": For finding shortest path in networking graph is used.	3. App":- For game trees decision the tree is used.
4) Two major types directed and undirected graph.	4) Two major type are binary tree and binary search tree.
5) More complex	5) less complex.

Binary Tree

where a parent node have atmost two child.



Binary Trees



Complete Binary Tree : Must have either two or zero child.

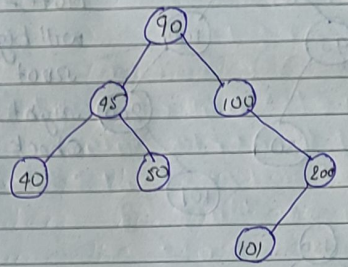
Ex. Imp

Binary Search Tree (BST)

Binary Search tree is a node-based binary tree data structure which has the following properties:-

- The left subtrees of a node contains only with keys lesser than the nodes key.
- The right subtree of a node contains only nodes with keys greater than the node's key.
- The left and right subtrees each must also be a binary search tree.

Note:- There must be no any duplicate nodes.



Binary Search Tree

Inorder (LR'R) :- 40, 45, 50, 90, 100, 101, 200
↳ It's also refer as Tree sort

preorder (R'LR) :- 90, 45, 40, 50, 100, 200, 101

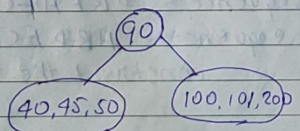
postorder (LRR') :- 40, 50, 45, 101, 200, 100, 90.

8) construct the B.S.T by following Inorder & preorder:-

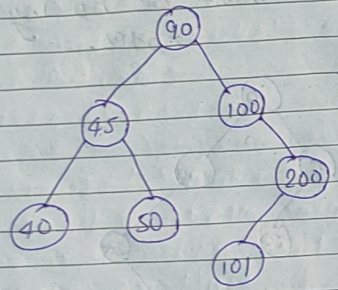
Inorder: 40, 45, 50, 90, 100, 101, 200

preorder: 90, 45, 40, 50, 100, 200, 101

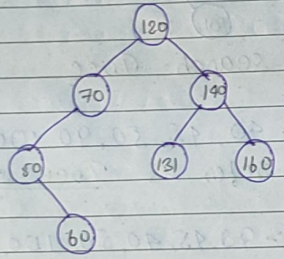
Step 1



Step2: In Preorder 45 comes at first position in 40, 45 & 50, 100 comes closed



Note:- if two no. will bigger than the root node for right side then check the preorder



B.S.T

Inorder (LR'R): 50, 60, 70, 120, 131, 140, 160

Preorder (RLR): 120, 70, 50, 60, 140, 131, 160

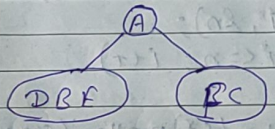
Postorder (LRR'): 60 50, 70, 131, 160, 140, 120

Q.) Let us consider the below traversals:-

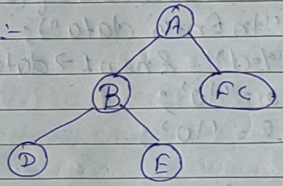
Inorder sequence:- DBEAFC
Preorder sequence:- ABDECF
construct the B.S.T

Solve) Here no any kind of numbering is given so we check first preorder then Inorder to check sequence.

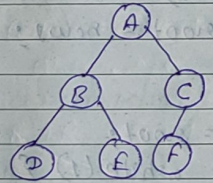
step1:-



step2:-



step3:-



Q.) W.A.P in C to create and display BST?

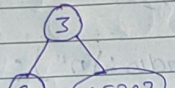
```
Struct node
{
    int data;
    Struct node *r, *l;
};
typedef struct node s;
    s* new1, *t, *root;
```

```
main()
{
  int i, n;        no. of
  printf("\n Enter nodes");
  scanf("%d", &n);
  for(i=1; i<=n; i++)
    {
      new1 = (s*) malloc (sizeof (s));
      printf("\n Enter data");
      scanf("%d", &new1->data);
      new1->r = '\0';
      new1->l = '\0';

      if(i==1)
          root = new1;
      else
        {
          t = root;
          while(1) {
            if(new1->data > t->data)
              {
                if(t->r == '\0')
                  {
                    t->r = new1;
                    break;
                  }
                else {
                  if(t->l == '\0')
                  {
                    t->l = new1;
                    break;
                  }
                else
                  {
                    t = t->l;
                  }
                }
              }
            }
        }
    }
  printf("\n Inordus : \n");
      inordus (root);

  Printf("\n preordus : \n");
    preordus (root);

  Printf("\n postordus : \n");
    postordus (root);
}
    void inordus (s *t)
  {
      if(t == '\0')
          return;
      inordus (t->l);
      printf("%d \t", t->data);
      inordus (t->r);
  }
    void preordus (s *t)
  {
      if(t == '\0')
          return;
      printf("%d \t", t->data);
      inordus (t->l);
      inordus (t->r);
  }
```

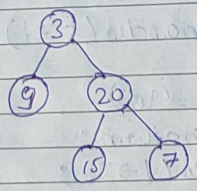
```
void postorder(struct t)
{
  if(t==NULL)
    return;
  inorder(t->l);
  inorder(t->r);
  printf("%d\t", t->data);
}
```

Q) Inorder [9 3 15 20 7]
preorder [3, 9, 20, 15, 7]

step1:



step2:

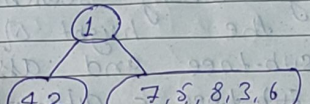


This became a tree because in the left child node of root is greater then the root node.

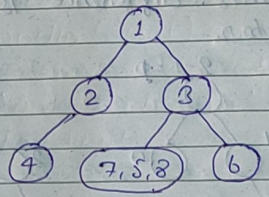
Note:- If a binary tree is given and the Inorder of this tree is sorted then its must be Binary search tree otherwise given tree is not BST.

Q) Inorder: {4,2,1,7,5,8,3,6}
preorder: {1,2,4,3,5,7,8,6}
construct Binary Tree.

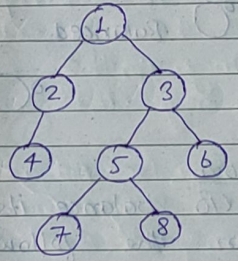
step 1:



step 2:

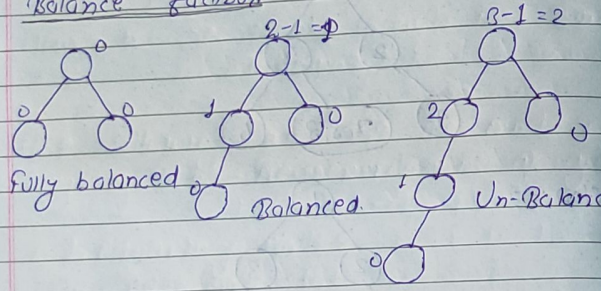


step 3:



AVL Tree (Height Balance Tree)

Named after their inventor Adelson Veiski and Landis AVL Trees are height balancing binary search tree. AVL tree checks the height of the left and right sub-tree and assures that the ~~right sub-tree~~ & difference is not more than 1. This difference is called the Balance factor.

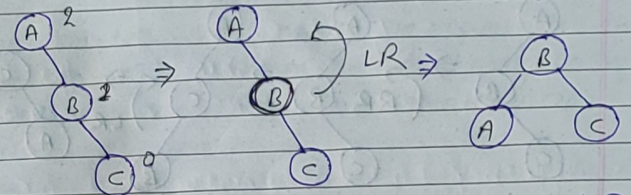


* AVL Rotation

To balance itself, an AVL tree may perform the following four kinds of rotations :-

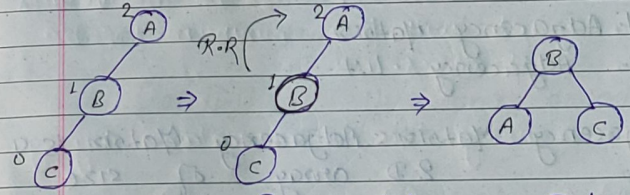
- Left rotation
- Right rotation
- Left-right rotation
- Right-left rotation.

1) Left rotation :- If a tree became unbalanced from the ~~left~~ right hand side when a node is inserted into the right subtree then left rotation will require



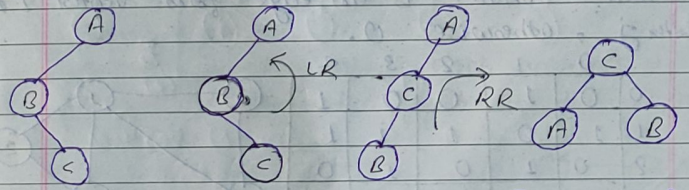
Right un-balanced left rotation Balanced Tree.

2) Right rotation :- If a tree became unbalanced from the left hand side when a node is inserted into the left subtree then right rotation will required.



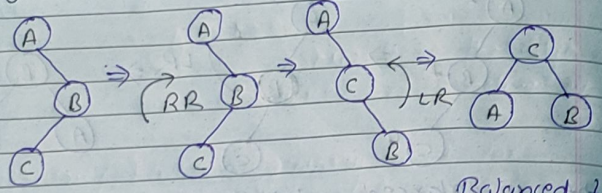
Left-unbalanced Right rotation Balanced tree.

3) Left-right Rotation :-



Balanced Tree

4) Right-left Rotation:-



Balanced tree

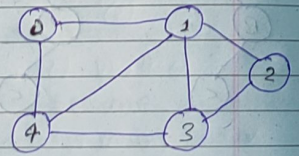
Q) How to represent Graph in Data structure?

Ans The following two are the most commonly used Representations of graph:-

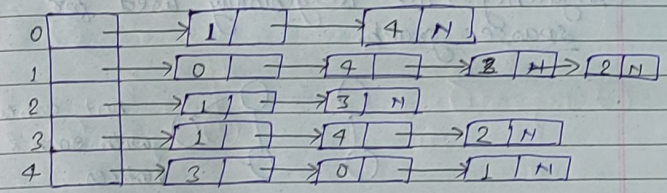
1. Adjacency Matrix
2. Adjacency List

1) Adjacency Matrix: Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j, otherwise 0.

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



2) Adjacency list:-
An array of lists is used. The size of the array is equal to the number of vertices. The weights of edges can be represented as lists of pairs.



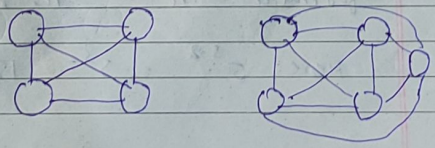
1) Adjacency Matrix:-
it is a matrix A[n][n] where n is no. of vertices &

{ a[i][j] = 1 if i and j are adjacent
= 0 otherwise }

time complexity = $O(n^2)$
where n = no. of vertices.

- And Adjacency Matrix is used for dense graph.

dense graph:- when each vertices is connected with all the vertices like

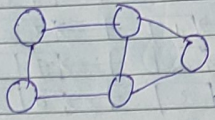


Time complexity of Adgacency list is

$= O(n+2e)$

where e is the edge which we use twice in undirected graph.

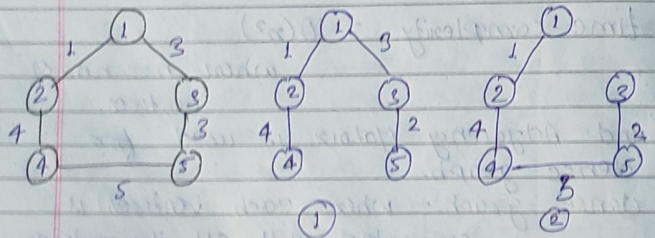
- Adgacency list is generally used for sparse graph.



not connected with each & every vertices.

* Spanning Tree:- A spanning tree is a tree that connects all the vertices of a graph with the minimum possible no. of edges.

Note:- Spanning tree never contains any cycle.



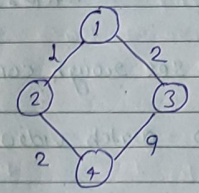
formula: $n^{(n-2)}$ n: no. of vertices

Here $5^{5-2} = 5^3 = 125$ spanning trees can be possible.

Note:- If a graph is a complete graph with n vertices then total number of spanning trees is $n^{(n-2)}$ where n is the number of nodes in the graph.

* Minimum spanning trees:- The minimum spanning trees is the tree whose sum of the to edges is minimum.

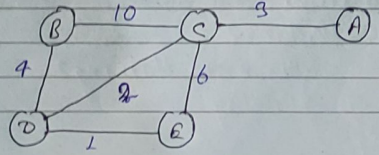
* weighted graph:- A weighted graph is a Graph in which each branch is given a numerical weight.

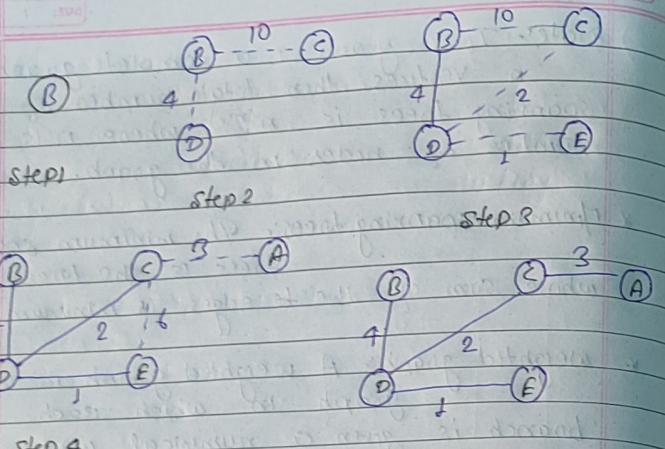


weighted graph.

Prim's Algorithm

prims algorithm starts with the single node and explore all the adjacent node with all the connecting edges at a every step. The edges with the minimal weights causing no cycle in the graph got selected.





Dijkstra's algo → see rough copy / video

floyd warshall algo → watch video

HASHING

Hashing is a technique or process of mapping keys, values into the hash table by using a hash function. It is done for faster access to element. The efficiency of mapping depends on the efficiency of the hash function used.

Let a hash function H(x) maps the value x at index x%.10 in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at position [1,2,3,4,5] in the array or hash table respectively.

A = [11, 12, 13, 14, 15] n = 5

H(x) = x %. 10

11 %. 10 = 1
12 %. 10 = 2
13 %. 10 = 3
14 %. 10 = 4
15 %. 10 = 5

1	11
2	12
3	13
4	14
5	15

Types of Hashing

1.) open Hashing (closed addressing)
↳ chaining

2) closed Hashing (open addressing)

- Linear Probing (searching)
- Quadratic Probe
- double Hashing.

ex:- A = 3, 2, 9, 6, 11, 13, 7, 12

$h(k) = 2k+m$

$m = 10$

where division method is $h(k_i) = k_i \% 10$

where m is the size of Hash table.

No key	location
3	$(2\times3+3)\%10 = 9$
2	$(2\times2+3)\%10 = 7$
9	1
6	5
11	5
13	9
7	7
12	7

Array

0	3	→ [ ][ ]
1	9	
2		
3		
4		
5	6	
6		
7	2	→ [7][N] → [12][N]
8		
9	3	→ [13][NULL]

Hash table

Imp Questions

→ what is Hashing?
→ what is collision in Hashing?
→ Types of Hashing.
→ chaining.

8) A = 3, 2, 9, 6, 11, 13, 7, 12

Use Division method and open addressing to store these values using linear Probing

$h(k) = 2k+3, \quad m = 10$

Key	Location	$k_i + m$	Probes
3	$(2\times3)+3\%10 =$	9	1
2	$((2\times2)+3)\%10 =$	7	1
9		1	1
6		5	1
11		5	2
13		9	2
7		7	2
12		7	6

Hash table

Array	
0	13
1	9
2	12
3	
4	
5	6
6	11
7	2
8	7
9	3