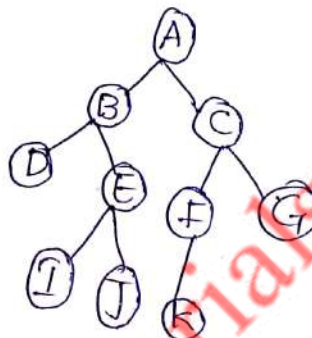


* A tree is a non-linear data structure in which the elements are organised in the form of hierarchical structure.

* A tree is a collection of elements (or) nodes and some associated operations.

* Hence the tree can also be called as an abstract data type.

Eg:-



Terminology of tree:-

1. Root:- In the tree, the topmost node can be called as Root (or) Root node.

* Every tree must contain only one root node.

2. Edge:- In the tree, a connecting link between any two nodes is known as an edge. If the tree contains 'n' no. of nodes then the tree contains $n-1$ no. of edges.

3. Parent:- In the tree, if a node contains child (or) children then such node can be called as a parent (or) parent node.

4. Child:- In the tree, if a node is having parent node then such node can be called as a child.

5. Leaf: In the tree, if a node does not have child (or) children then such node can be called as leaf (or) leaf node.

* leaf nodes can also be called as external nodes (or) terminal nodes.

6. Internal nodes: In the tree, except the leaf nodes the remaining nodes can be called as internal nodes.

* the internal nodes can also be called as non-terminal nodes.

7. Degree: In the tree, the total no. of childs of a particular node is said to be degree of that particular node.

* Among all the nodes the node which is having highest degree can be treated as the degree of the entire tree.

8. Height: In the tree, the total no. of edges from any leaf node to the specific node is said to be the height of that particular node.

* the height of the entire tree is the height of the root node.

* The height of the leaf nodes is 'zero'.

9. Depth: In the tree, the total no. of edges from the root node to any particular node is said to be depth of that node.

* The highest depth of a node can be treated as the depth of the tree.

* The depth of the root node is zero.

www.JntukMaterials.com

10. Level:- In the tree, usually the root node is represented at level 'zero' and its children are represented at level 'one' and their children will be represented at level 'two' and so...

11. Path:- The sequence of consecutive edges from one node to another node is known as a path.

* The length of the path is equal to the total no. of nodes existed along that particular path.

12. Sub tree:- In the tree, A subtree can be formed by using child (or) children of a particular node.

13. Ancestors & Descendants:- In the tree, all the nodes from root node to specific node are said to be ancestors for that node and all the nodes from the specific node to any leaf node are said to be descendants for that node.

14. Siblings:- In the tree, the nodes which are having the same parent node are said to be sibling.

General tree representation:-

Usually the general trees are represented using two ways they are 1. left child and 2. right sibling representation.

2.

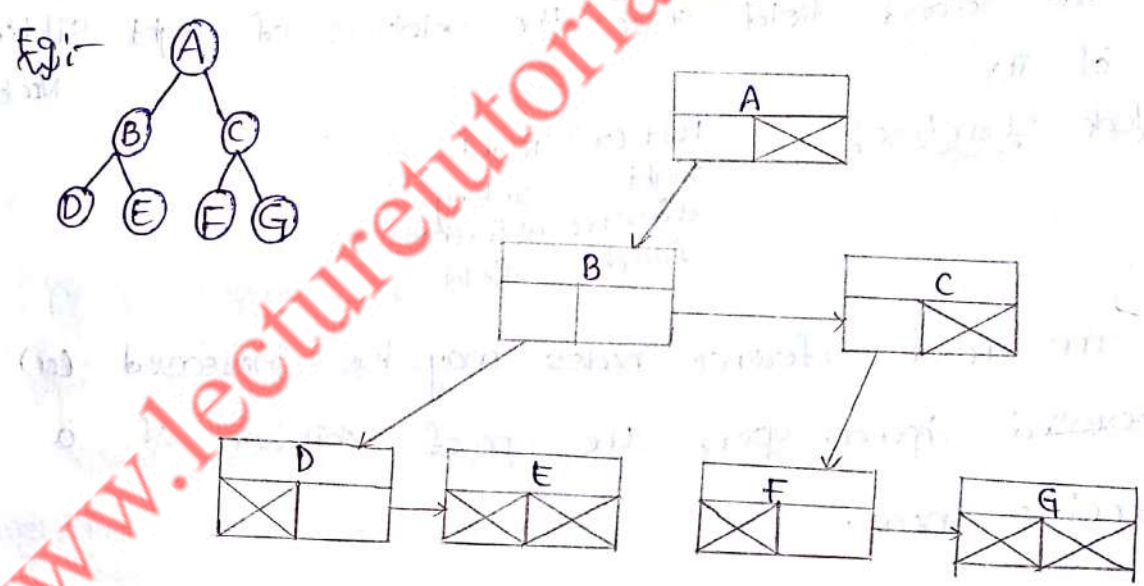
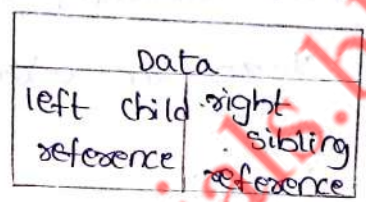
* In this representation, we use a special node which contains three fields such as data, left child reference and right sibling reference fields.

* The data field stores the actual element (or) value.

* The left child reference field stores the address of left child of the specific node

* The right sibling reference field stores the address of right sibling of the particular node.

Node structure:-



'X' indicates NULL.

In this representation we use two special nodes such as data node and reference node.

Data Node:- The data node contains 2 fields.

* The first field stores the actual element (or) value.
the second field stores the address of either left (or) right child.

Node structure:-

Data field	left (or) right child field
------------	-----------------------------

Reference Node:-

The reference node contains two fields
* The first field stores the address of either left (or) right child of a specific node.

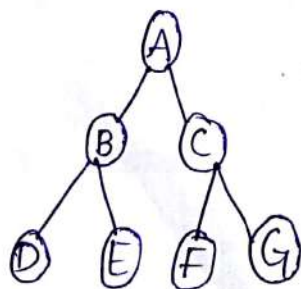
* The second field stores the address of right sibling of its

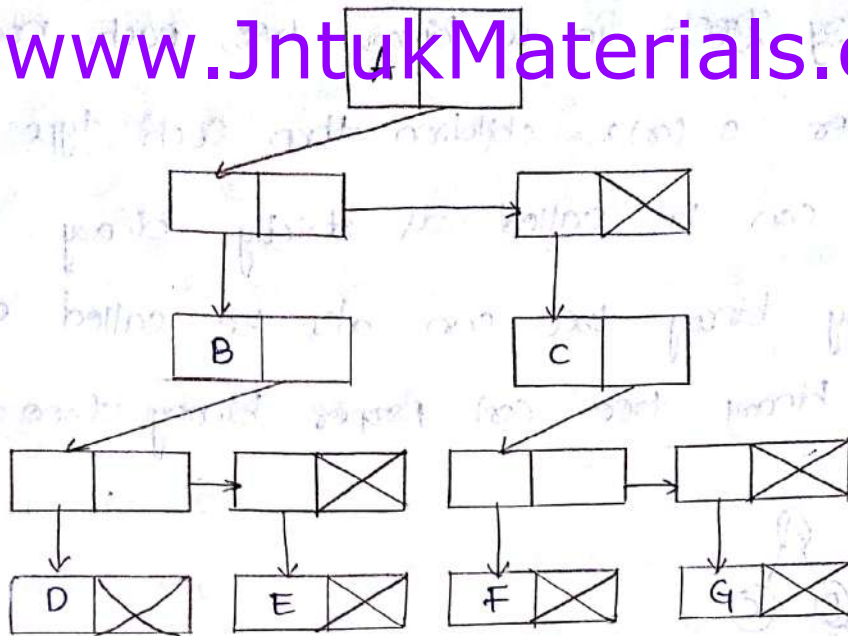
Node structure:-

left (or) right reference field	Right sibling reference field
---------------------------------	-------------------------------

* The no. of reference nodes may be increased (or) decreased depends upon the no. of children of a specific node.

Eg:-





Binary Trees:-

* In a tree, if every node contains either '0' (or) '1' (or) '2' children but not more than '2' children then such type of binary tree can be called as ^abinary tree

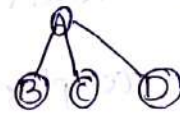
Eg:-

(a)

(b)

(c)

(d)



Not a binary tree
(e)

* The binary trees are widely used to represent algebraic expressions

Types of binary Trees:-

Based on the frequently usage the binary trees are classified into 5 types. they are.

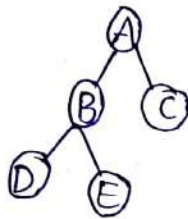
1. strictly binary tree
2. complete binary tree
3. Extended binary tree
4. Binary Search tree
5. threaded binary tree.

Strictly binary Tree:- In a binary tree, each node

contains either 0 (or) 2 children then such type of binary tree can be called as strictly binary tree.

* The strictly binary tree can also be called as either full binary tree (or) proper binary tree (or) 2-tree

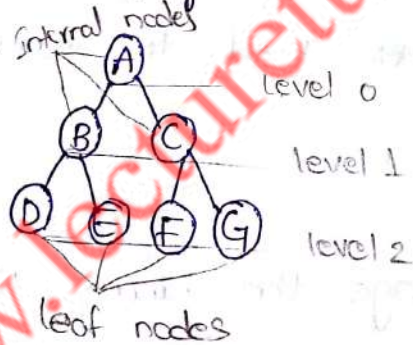
Eg:-



Complete Binary Tree:- In a binary tree, if every node contains exactly 2 children and all the leaf nodes are at same level then such type of binary tree can be called as a complete binary tree.

* A complete binary tree can also be called as a perfect binary tree.

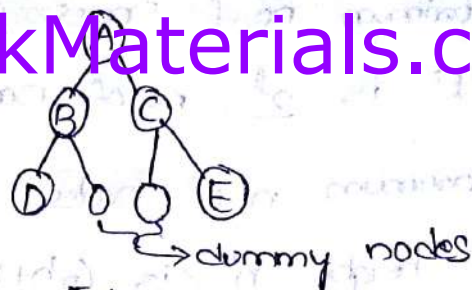
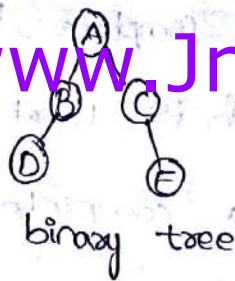
Eg:-



Extended binary tree:- If a binary tree is converted into strictly binary tree by adding some dummy nodes to the existing nodes then the converted strictly binary tree can be called as an extended binary tree.

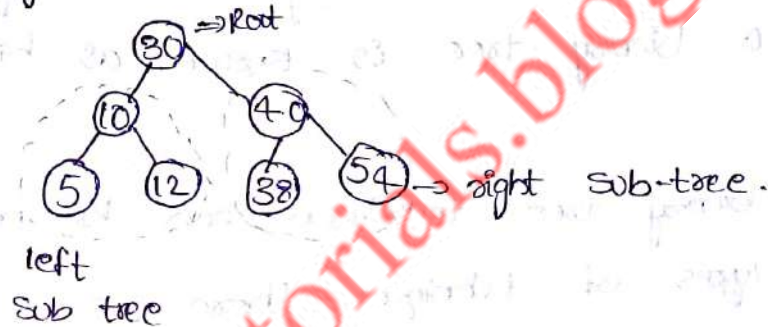
Eg:-

www.JntukMaterials.com



* Binary Search Tree:- In a binary tree, the values in of all the nodes in the left sub-tree are less than the root node and the values of all the nodes in the right sub-tree are greater than (or) equal to the root node then such type of binary tree can be called as a binary search tree.

Eg:-



Threaded Binary Tree:- When a binary tree is implemented when in the form of linked list then we may get NULL values in some of the fields of the nodes. If NULL values are replaced with some addresses of the nodes in that particular binary tree then such type of binary tree can be called as a threaded binary tree.

Binary Tree Properties:-

1. If the binary tree contains 'N' no. of nodes then it contains $N-1$ no. of edges.
2. In the binary tree, every node contains only one parent node except the root node.

3. The maximum no. of nodes of a complete binary tree at level 'l' is 2^l no. of nodes. (l is level number)

4. The maximum no. of nodes of a complete binary tree with height 'h' is $(2^{h+1} - 1)$ no. of nodes

5. The total no. of leaf nodes of a complete binary tree is $(n+1)/2$ no. of nodes (n = total no. of nodes)

6. The total no. of external nodes = total no. of internal nodes + 1.

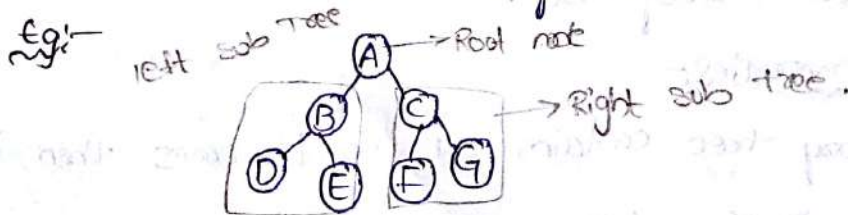
Binary Tree Traversal

* The process of visiting and displaying every node in a binary tree is known as binary tree traversal

* The binary tree traversal can be performed by using 3 types of techniques these are

- 1) In-ordered Traversal
- 2) Pre-ordered Traversal
- 3) Post-ordered Traversal.

In-ordered Traversal:- In this technique first the left sub-tree will be traversed (or) visited, and then root node and last the right sub tree will be traversed. In the binary tree.



In-ordered :- ~~traversal~~ D-B-E-A-F-C-G

procedure steps:-

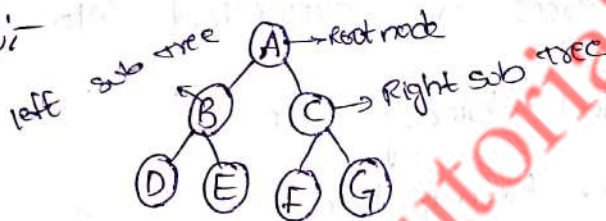
1. Traverse left sub-Tree
2. Traverse root node
3. Traverse right sub-Tree.

pre-ordered Traversal:- In this technique first the root node is visited and then left sub-Tree and last the right sub-Tree will be visited in the binary Tree.

procedure steps:-

1. Transverse root node
2. Transverse left sub-Tree
3. Transverse right sub-Tree.

Eg:-



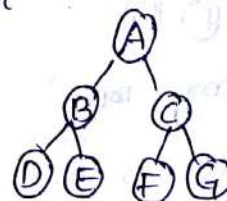
pre-order:- A - B - D - E - C - F - G

post-order Traversal:- In this technique first the left sub-Tree is will be traversed and then the right sub-Tree and last the root node will be traversed in the binary tree.

procedure steps:-

1. Transverse left sub-Tree
2. Transverse right sub-Tree
3. Transverse root-node.

Eg:-



pre-order:-

D - E - B - F - G - C - A

Threaded Binary Tree: If a binary tree is implemented in the form of double linked list then we may get NULL values in some fields of the nodes. If the NULL values are replaced with the addresses of some appropriate node in that binary tree then such binary tree can be called as a Threaded binary tree.

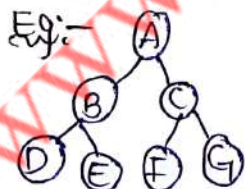
The NULL value in the left link field of a node is replaced with predecessor node address of that particular node in the inorder form. Similarly the NULL value in the right link field of a node is replaced with successor node address of that particular node in the inorder form.

The threaded binary trees are classified into two types

- one-way threaded binary tree
- two-way threaded binary tree.

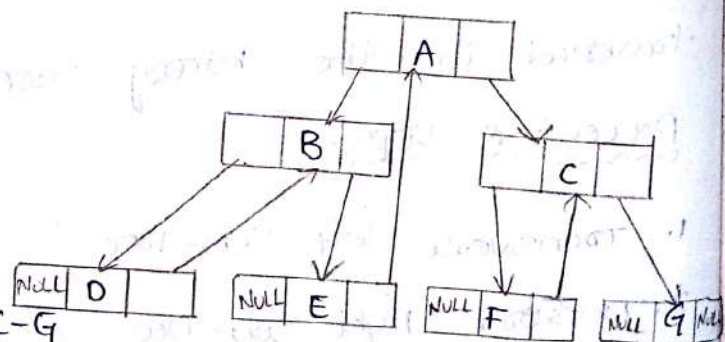
One-way Threaded Binary Tree:

In this only the NULL value stored in the right link field of a node is replaced with appropriate node address in the binary tree.



Binary Tree

In-order: D-B-E-A-F-C-G



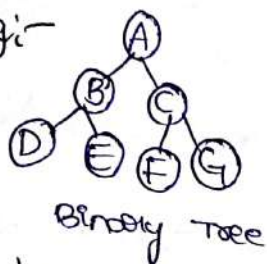
One-way threaded binary tree

* TWO-WAY Threaded Binary Tree

In this the NULL values stored in the left link

and right link fields of a node can be replaced with appropriate nodes addresses

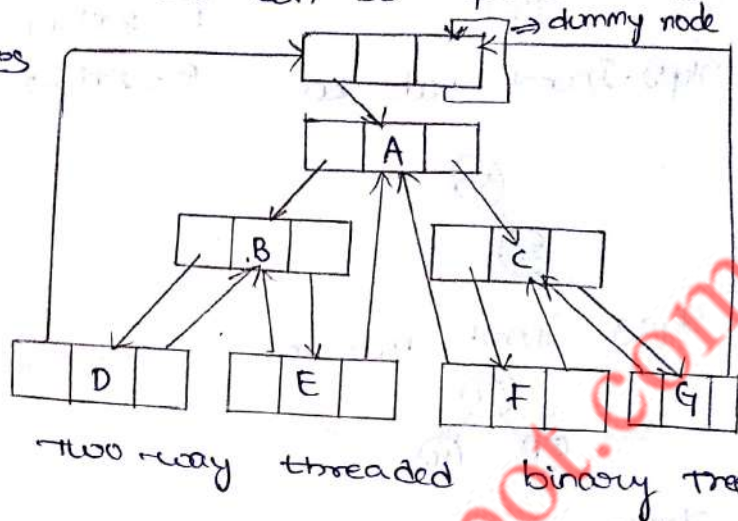
Eg:-



Binary tree

Inorder:-

D-B-E-A-F-C-G



Two-way threaded binary tree

Heap (or) Binary Heap:-

A heap is a tree based data structure in which the elements are arranged in the form of hierarchical structure

The heap follows two basic properties they are,

- i) shape (or) structure property
- ii) heap property

Shape property:- This property states that the heap must be either complete (or) almost complete binary tree.

In the complete binary tree the elements are organized from top to bottom and left to right in each level.

Heap property:- This property states that all the parent nodes in the heap must be either greater than (or) less than to their children.

* Usually the heaps are classified into two types they are

- i) max heap
- ii) Min heap.

Max heap:- In a heap if all the parent nodes are greater than their children then such heap can be called as Max heap.

Insertion
Eg:- construct a Max heap by the following elements.
50, 20, 10, 40, 100.

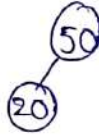
Step:1 - Insert node 50



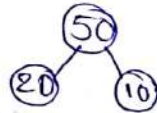
$$L = 2 \times i + 1 = 1$$

Step:2 - Insert node 20

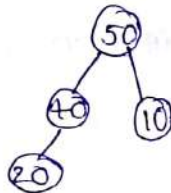
$$R = 2 \times i + 2$$



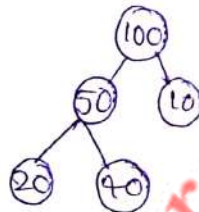
Step:3 Insert node 10



Step:4 - Insert node 40

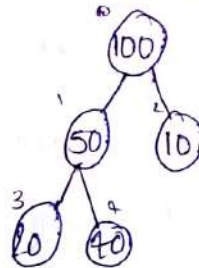


Step:5 Insert node 100

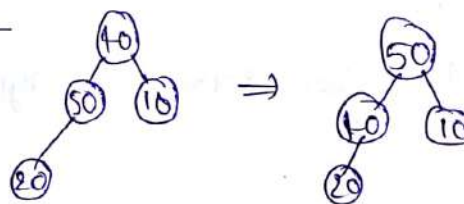


Deletion:-

Eg:- Delete the element from the above Max heap.



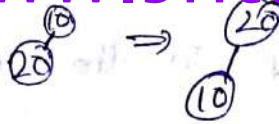
Step (1):-



(2) delete node 50



(3) delete node



4) delete node 20.

10

5) delete node 10

Min heap In a heap all the parent nodes are less than their children then such heap can be called as Min heap.

Priority Queue: A priority queue is a linear data structure

Priority Queue:- A priority queue is a linear data structure in which the elements are stored in the sequential order.

* A priority queue is a special queue in which the elements are stored in the order as they arrive along with their priority and the elements will be deleted depends upon either max priority (or) Min priority.

* Usually the priority queues can be used in CPU scheduling.

* The priority queue is classified into two types they are

1) Max priority queue

2) Min priority queue.

Max priority queue:- In this the elements are stored in the order as they arrive along with priorities and the elements are deleted depends upon highest priority.

Eg:- Insert the following elements into the max priority queue
elements are 40, 20, 30, 10 and priorities 5, 6, 4, 2 respectively.

pq[4]

	0	1	2	3
Data	40	20	30	10
priority	5	6	4	2

MAX priority queue

Max priority queue deleted

40	30	10
5	4	2

Min priority queue:- In this the elements are stored in the order as they arrive along with priorities and the elements are deleted depends upon lowest priority.

Eg:- Insert the following elements into the Min priority queue.

Min priority elements are 40, 20, 30, 10 and priorities 5, 6, 4, 2 respectively.

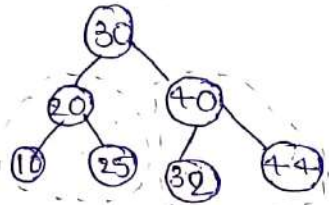
www.JntukMaterials.com

P9[4]

Data	40	20	30	10
priority	5	6	4	2

Binary search trees:- In a binary tree if the values of all the nodes in the left sub-tree are less than the root node and the values of all the nodes in the right sub tree are greater than (or) equal to the root node

Eg:-



On the Binary search tree we perform 5 basic operation they are

1. create()
2. Insert()
3. Delete()
4. Search()
5. Display()

Time complexity:- The binary search takes $O(\log n)$ time complexity while performing either search (or) insert (or) delete operation

create:- this operation creates an initial node in the binary search tree.

Eg:- 30

Insert:- this operation inserts a new node to the binary search tree

Eg:- Insert the following elements into the binary search tree.

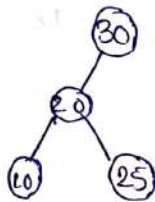
1. Insert node 20



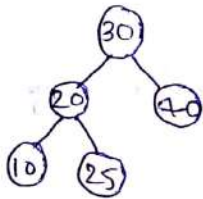
2. Insert node 25



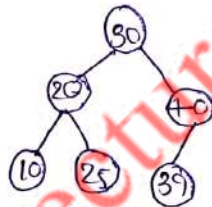
3. Insert node 10



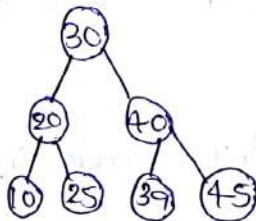
4. Insert node 40



5. Insert node 30



6. Insert node 45

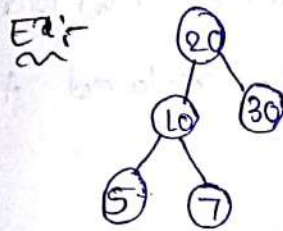


Delete :-

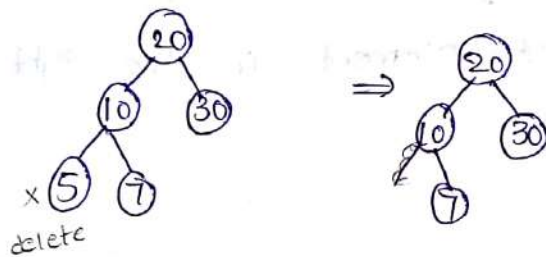
In the binary search tree the deletion can be performed in 3 possible cases they are.

- * case (1) deleting a leaf node
- case (2) deleting a node having only one child
- case (3) deleting a node

www.JntukMaterials.com
 In this case first we perform the search operation to find out the specific leaf node. and then the leaf node can be deleted by removing its connection from its parent node.



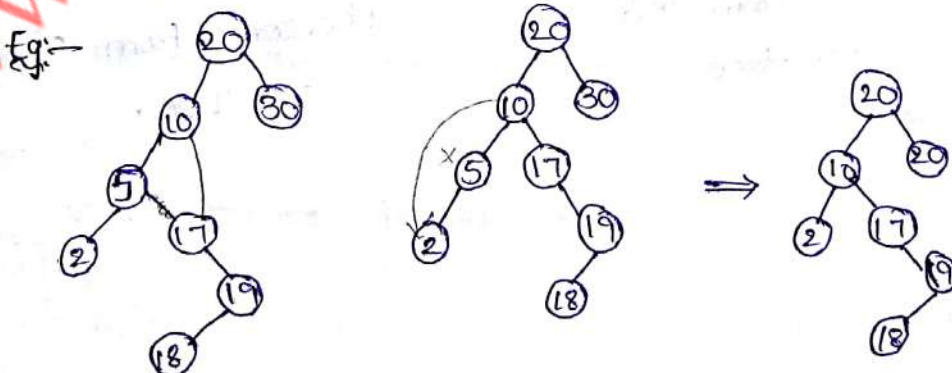
Delete node 5



case (2) :- deleting a node having only one child.

* In this case first we perform search operation to locate specific element for the deletion and then the connection will be established between child node and parent node of the specific node.

* In other words a node can be deleted having only one child by establishing the connection between its child node and parent node



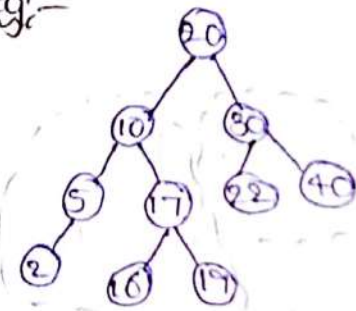
Delete node 5

Case (3): deleting a node having 2 children:-

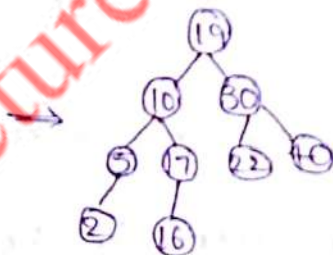
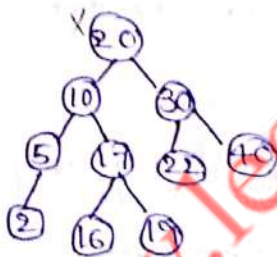
* In this case first we perform the search operation to locate the specific element for the deletion and then the highest element from the left sub-tree (or) lowest element from the right sub-tree can be replaced to the specific node for its deletion.

* In other words the node which is having 2 children can be deleted by replacing it with highest element in the left subtree (or) lowest element in the right subtree of that specific node.

Eg:-

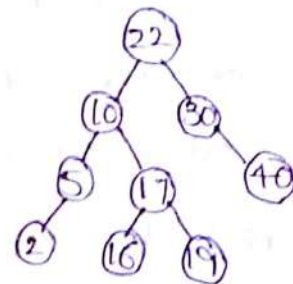


delete node 10



highest elements
chosen from left
sub-tree

(or)



less element is
chosen from right
sub-tree.

program to Implement binary Search tree

#include <iostream.h>

#include <conio.h>

struct node

{
int data;
node *left;
node *right;
}*root;

global structure.

class bstree.

{

public:

void create();

void insert();

void in-order(node *);

void pre-order(node *);

void post-order(node *);

void search(int);
};

void bstree::create()

{

root = new node;

cout << "Enter the data for the root node:";

cin >> root->data;

root->left = root->right = NULL;

}

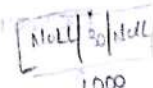
void bstree::insert()

{

int x;

node *temp, *parent;

cout << "Enter the element to be inserted:";




```
cin >> x;
temp = new node;
```

```
temp → data = x;
```

```
temp → left = NULL;
```

```
temp → right = NULL;
```

```
parent = NULL;
```

```
if (root == NULL)
```

```
{
    root = temp;
}
```

```
else
```

```
{
    node *ptr;
```

```
ptr = root;
```

```
while (ptr != NULL)
```

```
{
    parent = ptr;
```

```
if (x > ptr → data)
```

```
parent
```

```
ptr = ptr → right;
```

```
else
```

```
ptr = ptr → left;
```

```
}
```

```
if (x < parent → data)
```

```
parent → left = temp;
```

```
else
```

```
parent → right = temp;
```

```
}
```

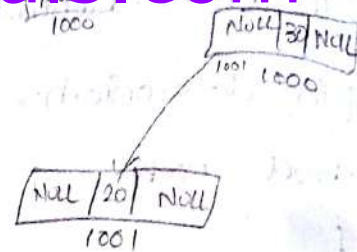
```
}
```

```
void bstree::in_order(node *ptr)
```

```
{
```

```
if (ptr != NULL)
```

```
{
```



in-order (ptr → left);

cout << " " << ptr → data << " ";

in-order (ptr → right);

}

}

void bstree::pre-order (node *ptr)

{

if (ptr != NULL)

{

cout << " " << ptr → data << " ";

pre-order (ptr → left);

pre-order (ptr → right);

}

}

void bstree::post-order (node *ptr)

{

if (ptr != NULL)

{

post-order (ptr → left);

post-order (ptr → right);

cout << " " << ptr → data << " ";

}

}

void bstree::search (int key)

{

int flag = 0;

node *temp;

temp = root;

if (temp == NULL)

{

cout << "\n BST is empty";


```
    }  
    else  
    {  
        while(temp != NULL)  
        {  
            if (temp->data == key)  
            {  
                cout << key << " " << "is found" << endl;  
                flag = 1;  
                break;  
            }  
            if (temp->data > key)  
                temp = temp->left;  
            if (temp->data < key)  
                temp = temp->right;  
        }  
        if (flag == 0)  
            cout << key << " " << "is not found" << endl;  
    }  
}  
void main()  
{  
    btree bs;  
    int option, k;  
    clrscr();  
    cout << "1. create() " << endl;  
    cout << "2. insert() " << endl;  
    cout << "3. in_order() " << endl;  
    cout << "4. pre_order() " << endl;  
    cout << "5. post_order() " << endl;  
}
```

```
cout << "(6. Search)" << endl;
cout << "7. Exit" << endl;
```

```
do
{
```

```
cout << "Enter your option: ";
```

```
cin >> option;
```

```
switch (option)
```

```
{
```

```
case 1:
```

```
bs.create();
```

```
break;
```

```
case 2:
```

```
bs.insert();
```

```
break;
```

```
case 3:
```

```
bs.in_order(root);
```

```
break;
```

```
case 4:
```

```
bs.pre_order(root);
```

```
break;
```

```
case 5:
```

```
bs.post_order(root);
```

```
break;
```

```
case 6:
```

```
cout << "Enter the element to be searched: ";
```

```
cin >> k;
```

```
bs.search(k);
```

```
break;
```

```
default:
```

```
cout << "Enter a valid option: ";
```

```
}
```

```
}
```

```
while (option != 7);
```

```
getch();
```

```
}
```