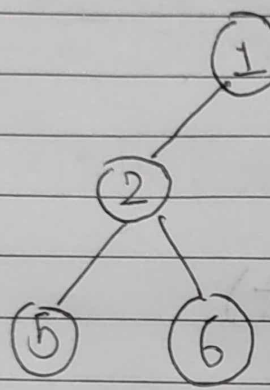
Harsh Kaliwal
2003085
C21

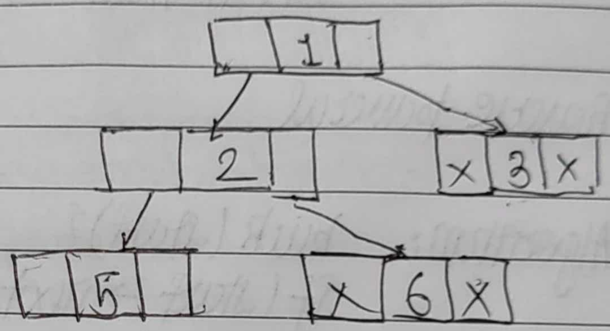
Experiment 8

Aim: Implement Binary search tree.

Theory:

Binary tree means that the node can have maximum two children. 'Binary' itself suggests that 'two' therefore each node can have either 0, 1 or 2 children





Properties:

P1: Minimum number of nodes in a binary tree of height H. $= H+1$.

P2: Maximum number of nodes in a binary tree of height H $= 2^{H+1} - 1$.

P3: Maximum number of nodes at any level 'L' in a binary tree $= 2^L$.

P4: Total number of leaf nodes in a Binary Tree = Total number of nodes with 2 children + 1.

1. Strict Binary Trees: Strict binary trees are those binary trees whose nodes either have 2 or 0 children.

2. Complete Binary Trees: Complete Binary Trees are those that have all their different levels completely filled. The only exception to this could be their last level, whose keys are predominantly on the left.

3. Perfect Binary trees: ~~Trees to~~ Binary trees whose leaves are present at the same level and whose internal nodes carry two children.

4. Balanced Binary trees: It is Binary tree in which height of the left and the right sub-trees of every node may differ by at most 1.

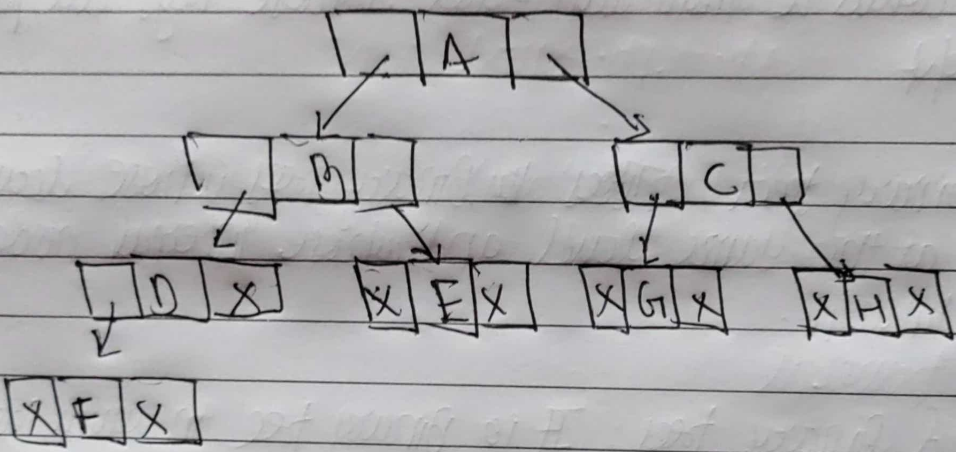
Representation:

1. Using Array: The array representation stores the tree data by scanning elements using level order fashion. So it stores nodes level by level. If some element is missing it left blank spaces for it.

1	2	3	4	5	6	7
5	10	17	-	8	15	25

2.

2. Using linked list : We use a double linked list to represent a binary tree. In this, every node consists of three fields. First field for storing left child address & 2nd for storing actual data & 3rd for storing right child address.



PROGRAM:

Write a menu driven code to implement BinarySearch Tree.

Code:

```
#include <iostream>
#include <conio.h>
using namespace std;

struct Node
{
    int data;
    Node *left;
    Node *right;
};

void display(Node *root)
{
    if (root == NULL)
        return;
    cout<<" "<<root->data;
    display(root->left);
    display(root->right);
}

Node *minValueNode(Node *root)
{
    Node *current = root;
    while (current->left != NULL)
    {
        current = current->left;
    }
    return current;
}

Node *maxValueNode(Node *root)
{
    Node *current = root;
    while (current->right != NULL)
    {
        current = current->right;
    }
    return current;
}

Node *insert(Node *root, int data)
{
    if (root == NULL)
```

```
    {
        root = (Node *)malloc(sizeof(Node));
        root->data = data;
        root->left = NULL;
        root->right = NULL;
    }
    else if (data <= root->data)
    {
        root->left = insert(root->left, data);
    }
    else
    {
        root->right = insert(root->right, data);
    }
    return root;
}

Node *deleteNode(Node *root, int data)
{
    if (root == NULL)
    {
        return root;
    }
    else if (data < root->data)
    {
        root->left = deleteNode(root->left, data);
    }
    else if (data > root->data)
    {
        root->right = deleteNode(root->right, data);
    }
    else
    {
        if (root->left == NULL && root->right == NULL)
        {
            free(root);
            root = NULL;
        }
        else if (root->left == NULL)
        {
            Node *temp = root;
            root = root->right;
            free(temp);
        }
        else if (root->right == NULL)
        {
            Node *temp = root;
            root = root->left;
```

```
            free(temp);
        }
        else
        {
            Node *temp = minValueNode(root->right);
            root->data = temp->data;
            root->right = deleteNode(root->right, temp->data);
        }
    }
    return root;
}

void search(Node *root, int data)
{
    if (root == NULL)
    {
        cout<<"Not found\n";
        return;
    }
    else if (data < root->data)
    {
        search(root->left, data);
    }
    else if (data > root->data)
    {
        search(root->right, data);
    }
    else
    {
        cout<<"Found "<<root->data<<endl;
    }
}

void postorder(Node *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        cout<<" "<<root->data;
    }
}

void inorder(Node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
```

```
        cout<<" "<<root->data;
        inorder(root->right);
    }
}

void preorder(Node *root)
{
    if (root != NULL)
    {
        cout<<" "<<root->data;
        preorder(root->left);
        preorder(root->right);
    }
}

int height(Node *root)
{
    if (root == NULL)
    {
        return 0;
    }
    else
    {
        int lheight = height(root->left);
        int rheight = height(root->right);
        if (lheight > rheight)
        {
            return (lheight + 1);
        }
        else
        {
            return (rheight + 1);
        }
    }
}

void deleteTree(Node *root)
{
    if (root != NULL)
    {
        deleteTree(root->left);
        deleteTree(root->right);
        free(root);
    }
}

void mirror(Node *root)
{
```

```
    if (root != NULL)
    {
        Node *temp = root->left;
        root->left = root->right;
        root->right = temp;
        mirror(root->left);
        mirror(root->right);
    }
}

int countNodes(Node *root)
{
    if (root == NULL)
    {
        return 0;
    }
    else
    {
        return (countNodes(root->left) + countNodes(root->right) + 1);
    }
}

int main()
{
    char ch;
    Node *root = NULL;
    int choice, data;
    while (1)
    {
        cout<<"\n1. Insertion\n2. Deleting a node\n3. Search\n4. Preorder Traversal\n5. Inorder Traversal\n6. Postorder Traversal\n7. Height of a tree\n8. Mirror of BST\n9. Count Total Numbers of Nodes\n10. Delete entire Tree\n11. Display\n12. Smallest Element in the Tree\n13. Largest Element in the Tree\n14. Exit\n\nEnter your choice: ";
        cin>>choice;
        switch (choice)
        {
        case 1:
            cout<<"\nEnter the data: ";
            cin>>data;
            root = insert(root, data);
            break;
        case 2:
            cout<<"\nEnter the data: ";
            cin>>data;
            root = deleteNode(root, data);
            break;
        case 3:
```

```cpp
            cout<<"\nEnter the data: ";
            cin>>data;
            search(root, data);
            break;
        case 4:
            cout<<"\nPreorder traversal: ";
            preorder(root);
            break;
        case 5:
            cout<<"\nInorder traversal: ";
            inorder(root);
            break;
        case 6:
            cout<<"\nPostorder traversal: ";
            postorder(root);
            break;
        case 7:
            cout<<"\nHeight: "<<height(root)<<endl;
            break;
        case 8:
            cout<<"\nMirror of tree: ";
            mirror(root);
            inorder(root);
            break;
        case 9:
            cout<<"\nCount Nodes: "<<countNodes(root)<<endl;
            break;
        case 10:
            cout<<"\nAre you sure you want to delete the tree?\n";
            cin>>ch;
            if (ch == 'y' || ch == 'Y')
                deleteTree(root);
            else
                cout<<"\nTree not deleted\n";
            break;
        case 11:
            display(root);
            break;
        case 12:
            cout<<"\nMinimum value: "<<maxValueNode(root)->data<<endl;
            break;
        case 13:
            cout<<"\nMaximum value: "<<minValueNode(root)->data<<endl;
            break;
        case 14:
            exit(0);
        default:
            cout<<"\nWrong choice.\n";
```

```
        }
    }
    return 0;
}
```

OUTPUT:

```
Try the new cross-platform PowerShell https://aka.ms/pscore6
PS D:\Harsh\SEM 3\DS\CODES> cd "d:\Harsh\SEM 3\DS\CODES\" ; if ($?) { g++ Tree.cpp -o 

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 1

Enter the data: 12

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 1

Enter the data: 13

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
```

```
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 1

Enter the data: 14

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 11
 12 13 14
1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit
```

```
Enter your choice: 2

Enter the data: 11

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 4

Preorder traversal:  12 13 14
1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 5

Inorder traversal:  12 13 14
1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
```

```
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 12

Minimum value: 14

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit

Enter your choice: 13

Maximum value: 12

1. Insertion
2. Deleting a node
3. Search
4. Preorder Traversal
5. Inorder Traversal
6. Postorder Traversal
7. Height of a tree
8. Mirror of BST
9. Count Total Numbers of Nodes
10. Delete entire Tree
11. Display
12. Smallest Element in the Tree
13. Largest Element in the Tree
14. Exit
```