Assignment No: 7

Aim:-

Implement binary tree using linked list and perform recursive traversals.

Theory:-

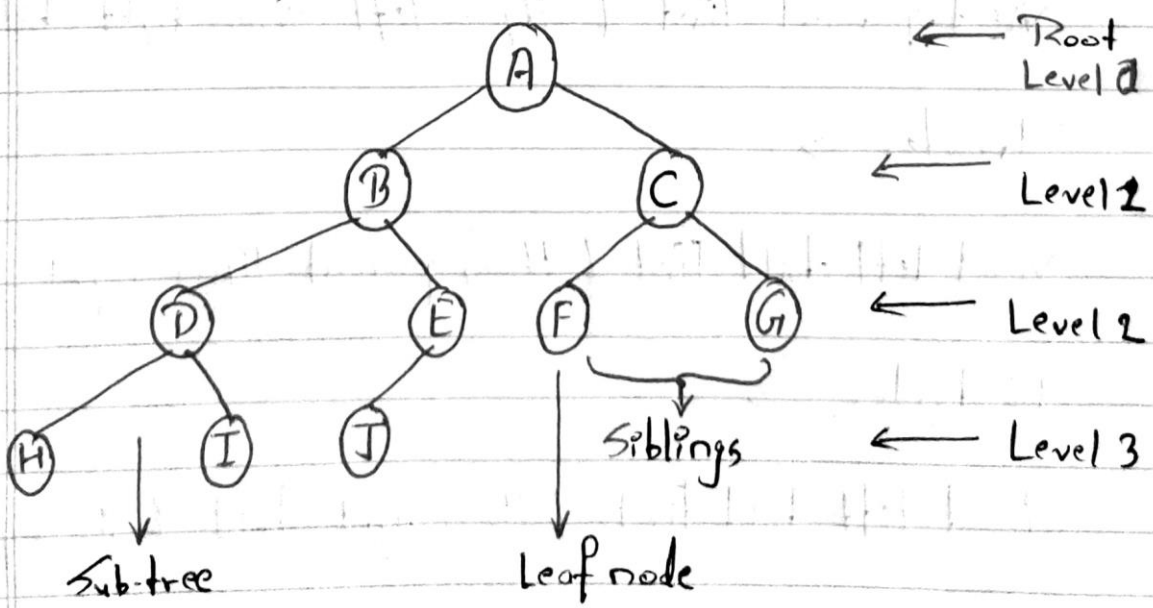
Tree ⇒

It represents the node connect by edge also a class of graphs that is acyclic is termed as trees.

Binary Tree ⇒

It is made up of nodes contains a 'left' reference, 'right' reference.

Every node in a tree is connected by directed edge from exactly one other node. This node is called parent. On other hand, each node can be connected to arbitary number of nodes, called children. Nodes with same parent are called siblings.



Insert Operation ⇒

The very first insertion creates the tree. Afterwards, whenever an element is to be inserted, first locate its proper location. Start searching from root node, then if data is less than key value, search for empty location.

Traversals ⇒

It is a process that visits all the nodes in tree. Since, a tree is non-linear data, there is no unique traversal.

Depth first Traversal
Breadth first Traversal

There are 3 different types of depth-first traversal-

Preorder ⇒

Visit parent first & then left then right children.

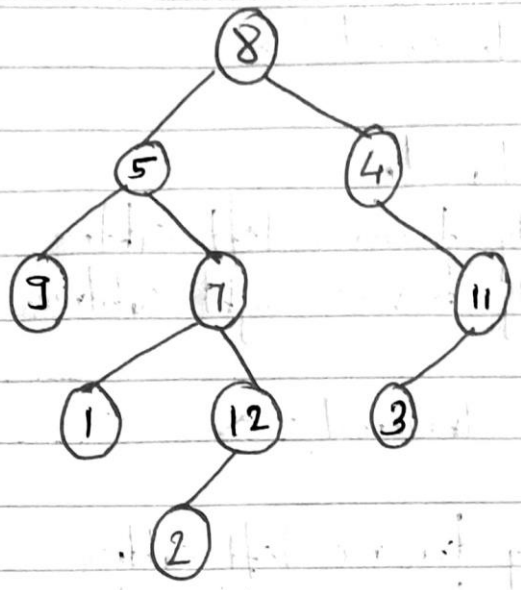
Inorder ⇒

Visit left child first & then parent then right child.

Postorder ⇒

Visit left child first & then right child & then parent.

For e.g:-



Postorder:-
9, 1, 2, 12, 7, 5, 3, 11, 4, 8

Preorder:-
8, 5, 9, 7, 1, 12, 2, 4, 11, 3

Level order:-
8, 5, 4, 9, 7, 11, 1, 12, 3, 2

Inorder:-
9, 5, 1, 7, 2, 12, 8, 4, 3, 11

Algorithm ⇒

Insert a node ⇒

Step 1: Search a node whose child node is to be inserted. This is a node at some level i if a node is to be inserted at level i+1 as either its left child or right child.

Step 2: Link a new node to the node that become parent node.

Inorder Traversal ⇒

Step 1: Recursively traverse left subtree.
Step 2: Visit root node
Step 3: Recursively traverse right subtree.

Preorder Traversal ⇒

Step 1: Visit root node
Step 2: Recursively traverse left subtree.
Step 3: Recursively traverse right subtree.

Postorder Traversal ⇒

Step 1: Recursively traverse left subtree
Step 2: Recursively traverse right subtree
Step 3: Visit root node

To copy one tree to another ⇒

```
Step 1: if (root == NULL)
            return NULL
Step 2: temp = new TreeNode
Step 3: temp → lchild = Tree Copy (root → lchild);
Step 4: temp → rchild = Tree Copy (root → rchild);
Step 5: temp → data = return.
```

Outcome ⇒

Learn OOP's features, understand & implement different operations on tree & binary tree.

Conclusion ⇒

Thus, we have studied the implementation of various binary tree operations.

Program Code :-

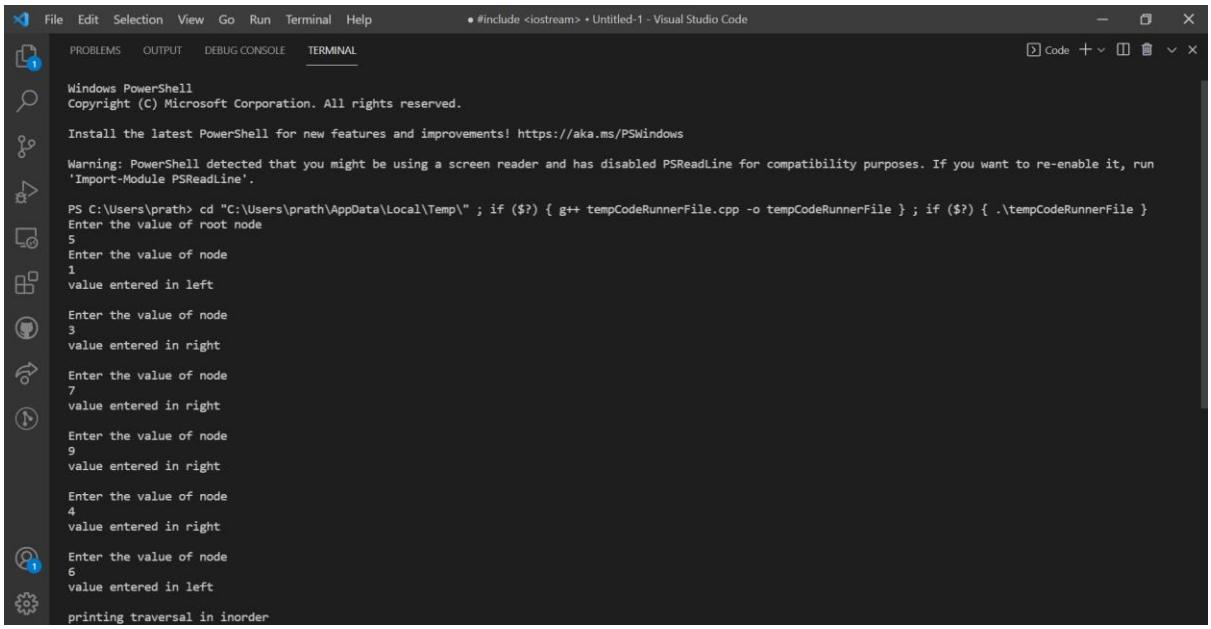
```
#include <iostream>
#include <conio.h>
using namespace std;
struct tree
{
   tree *l, *r;
   int data;
} *root = NULL, *p = NULL, *np = NULL, *q;

void create()
{
   int value, c = 0;
   while (c < 7)
   {
      if (root == NULL)
      {
         root = new tree;
         cout << "Enter the value of root node\n";
         cin >> root->data;
         root->r = NULL;
         root->l = NULL;
      }
      else
      {
         p = root;
         cout << "Enter the value of node\n";
         cin >> value;
         while (true)
```

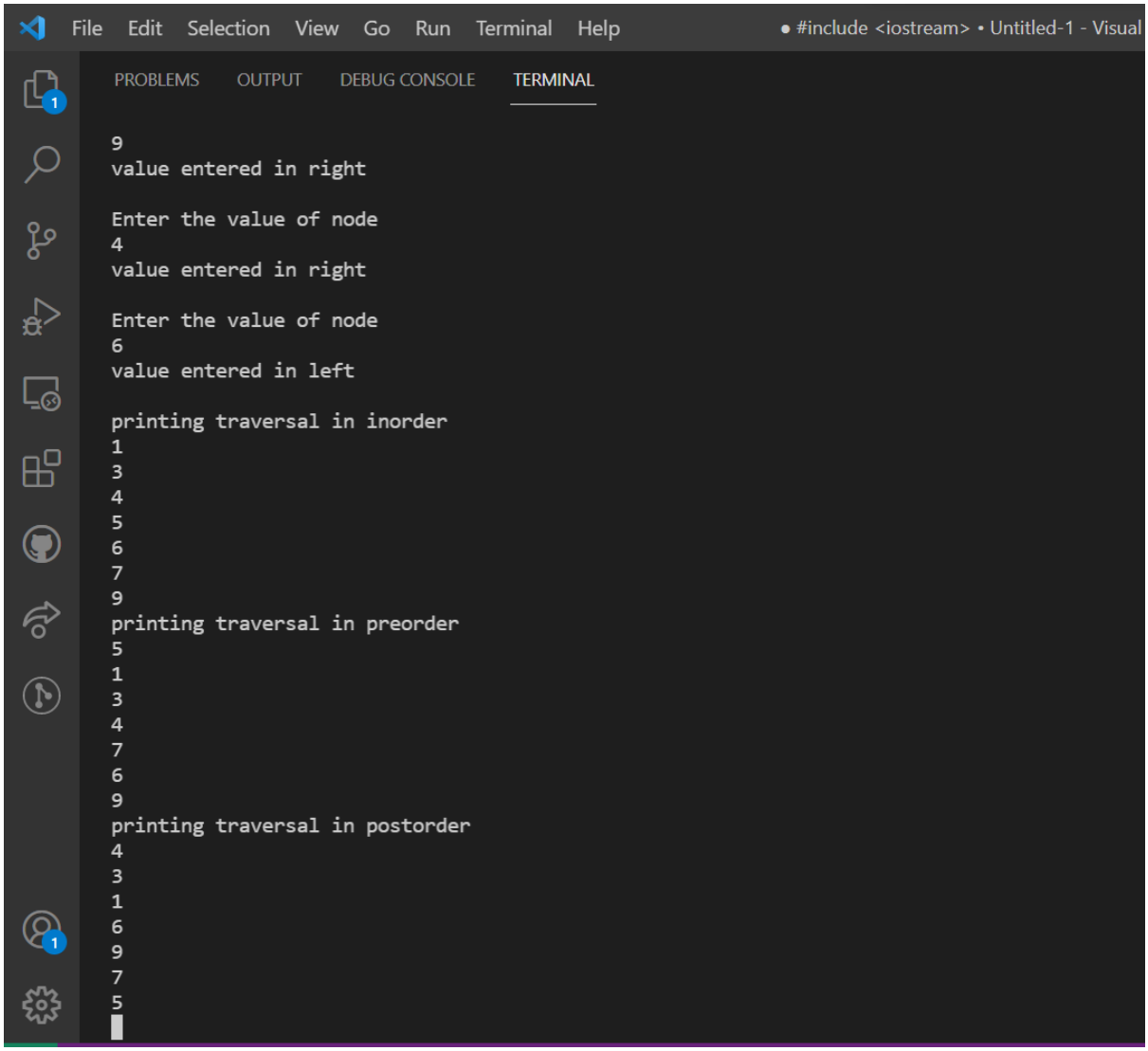
```
{
    if (value < p->data)
    {
        if (p->l == NULL)
        {
            p->l = new tree;
            p = p->l;
            p->data = value;
            p->l = NULL;
            p->r = NULL;
            cout << "value entered in left\n"<<endl;
            break;
        }
        else if (p->l != NULL)
        {
            p = p->l;
        }
    }
    else if (value > p->data)
    {
        if (p->r == NULL)
        {
            p->r = new tree;
            p = p->r;
            p->data = value;
            p->l = NULL;
            p->r = NULL;
            cout << "value entered in right\n"<<endl;
            break;
        }
```

```
                else if (p->r != NULL)
                {
                    p = p->r;
                }
            }
        }
    }
    c++;
  }
}
void inorder(tree *p)
{
  if (p != NULL)
  {
    inorder(p->l);
    cout << p->data << endl;
    inorder(p->r);
  }
}
void preorder(tree *p)
{
  if (p != NULL)
  {
    cout << p->data << endl;
    preorder(p->l);
    preorder(p->r);
  }
}
void postorder(tree *p)
{
```

```
    if (p != NULL)
    {
        postorder(p->l);
        postorder(p->r);
        cout << p->data << endl;
    }
}
int main()
{
    create();
    cout << "printing traversal in inorder\n";
    inorder(root);
    cout << "printing traversal in preorder\n";
    preorder(root);
    cout << "printing traversal in postorder\n";
    postorder(root);
    getch();
}
```

Program Output :-




File Edit Selection View Go Run Terminal Help
• #include <iostream> • Untitled-1 - Visual
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
9
value entered in right
Enter the value of node
4
value entered in right
Enter the value of node
6
value entered in left
printing traversal in inorder
1
3
4
5
6
7
9
printing traversal in preorder
5
1
3
4
7
6
9
printing traversal in postorder
4
3
1
6
9
7
5