

## WEEK 9 – LAB-B

1)

```
#include <iostream>
#include<queue>
using namespace std;
struct node
{
    int data;
    node *left;
    node *right;
};
int a[]={1,2,3,4,5,6,7,0,8,9,0,0,0,10,11,12,13,12,13,0,14,0,0,0,0,0,0,15,0,16,17};
struct node* insertion(int n)
{
    node *temp=new node;
    temp=NULL;
    if(n>sizeof(a)/sizeof(int))
    {
        return temp;
    }
    else if(!a[n])
    {
        return temp;
    }
    else if(a[n])
    {
        temp=new node;
        temp->data=a[n];
        temp->left=insertion(2*n+1);
        temp->right=insertion(2*n+2);
    }
}

void postorder(node *root)
{
}
```

```

    if(root==NULL)
        return;
    else
    {
        postorder(root->left);
        postorder(root->right);
        cout<<root->data<<" ";
    }
}

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

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

void levelorder(node *root)
{
    if (root == NULL) return;
    queue < node * > q;
    q.push(root);

    while (q.empty() == false) {
        node * node = q.front();
        cout << node -> data << " ";
        q.pop();
        if (node -> left != NULL)
            q.push(node -> left);
        if (node -> right != NULL)
            q.push(node -> right);
    }
}

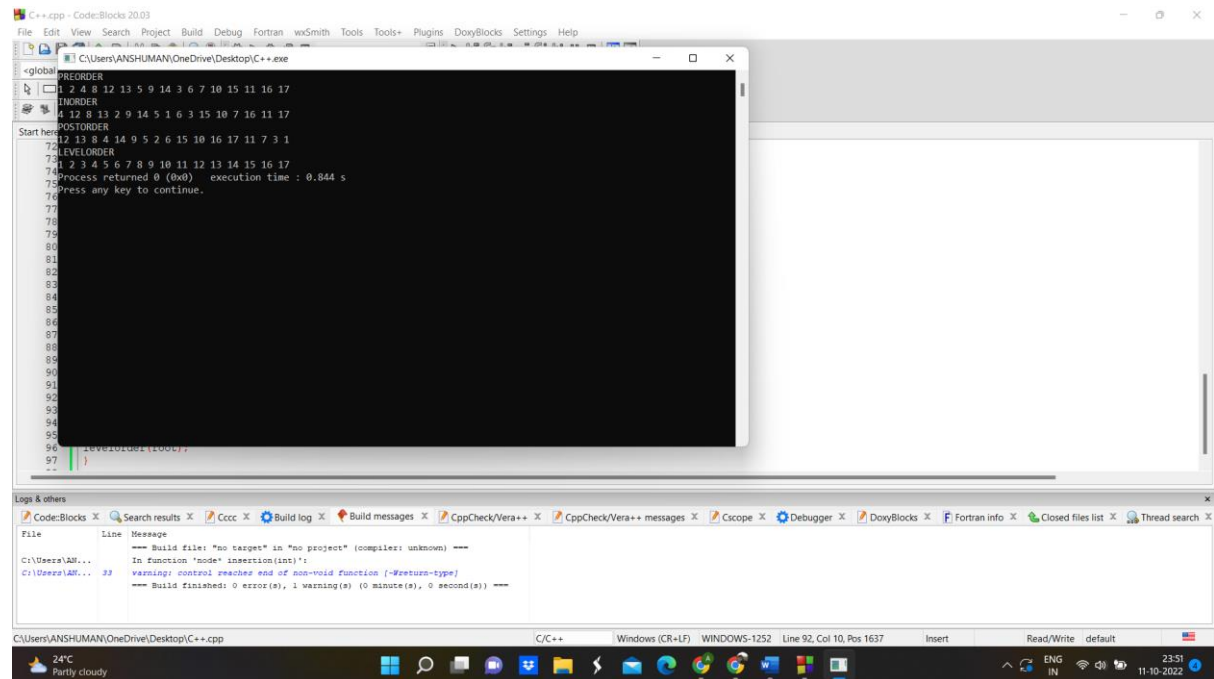
int main() {
    node *root;
    root=insertion(0);

```

```

preorder(root);
cout<<endl;
inorder(root);
cout<<endl;
postorder(root);
cout<<endl;
levelorder(root);
}

```



2)

```
#include <iostream>
```

```
#include<vector>
```

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
struct node
```

```
{
```

```
    int data;
```

```
    node *left;
```

```
    node *right;
```

```
};
```

```
int maxsum=0;
```

```
int a[]={1,2,3,4,5,6,7,0,8,9,0,0,0,10,11,12,13,12,13,0,14,0,0,0,0,0,0,15,0,16,17};
```

```
struct node* insertion(int n)
```

```
{
```

```
    node *temp=new node;
```

```
    temp=NULL;
```

```
    if(n>sizeof(a)/sizeof(int))
```

```
    {
```

```
        return temp;
```

```
    }
```

```
    else if(!a[n])
```

```
    {
```

```
        return temp;
```

```
    }
```

```
    else if(a[n])
```

```
    {
```

```
        temp=new node;
```

```

temp->data=a[n];
temp->left=insertion(2*n+1);
temp->right=insertion(2*n+2);

}
}
vector<int> longestPath(node* root)
{

if (root == NULL) {
    vector<int> temp
        = {};
    return temp;
}

vector<int> rightvect
    = longestPath(root->right);

vector<int> leftvect
    = longestPath(root->left);

if (leftvect.size() > rightvect.size())
    leftvect.push_back(root->data);

else
    rightvect.push_back(root->data);

return (accumulate(leftvect.begin(),leftvect.end(),0)
>accumulate(rightvect.begin(),rightvect.end(),0)
    ? leftvect

```

```

        : rightvect);
    }

int main() {
    node *root;
    root=insertion(0);
    vector<int> a=longestPath(root);
    for (int i = 0; i < a.size(); i++)
        cout << a[i] << " ";

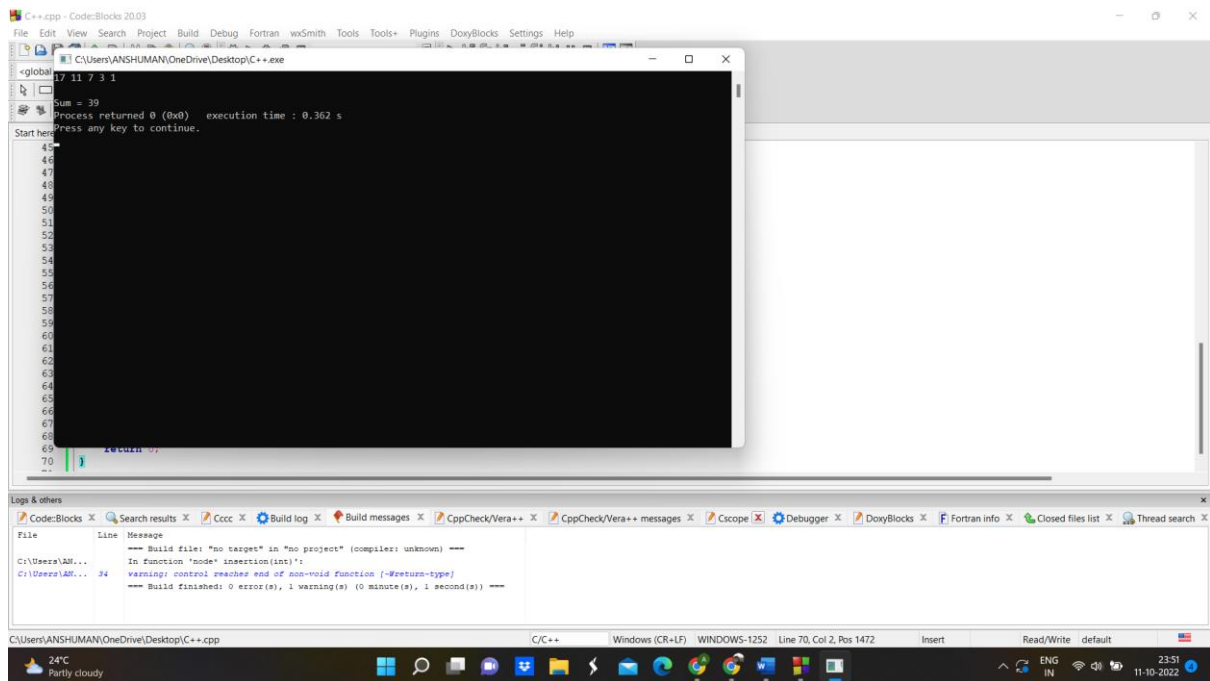
    cout << endl;

    cout << "\nSum = "

        << accumulate(a.begin(), a.end(), 0);

    return 0;
}

```



3)

```
#include <iostream>
#include<queue>
#include<stack>
#include <bits/stdc++.h>
using namespace std;
struct node
{
    char data;
    node *left;
    node *right;
};
struct node * newNode(char F)
{
    node *N=new node;
    N->left=NULL;
    N->right=NULL;
    N->data=F;
    return N;
}
node *insertnode(char c,queue<node *>&q,struct node *root)
{
    node * Node=newNode(c);
    if(root==NULL)
    {
        root=Node;
    }
    else if(q.front()->left==NULL)
        q.front()->left=Node;
    else
    {
        q.front()->right=Node;
        q.pop();
    }
    q.push(Node);
    return root;
}
node * createtree()
{
    node *root=NULL;
    char arr[9]={'A','B','C','D','E','F','G','H','I'};
    queue<node *>q;
    for(int i=0;i<9;i++)
        root=insertnode(arr[i],q,root);
    return root;
}
void levelorder(node *root)
{

```

```

if(root==NULL)
    return;
queue<node *>q1;
q1.push(root);
while(q1.empty()==false)
{
    node *f=q1.front();
    cout<<f->data<<" ";
    q1.pop();
    if(f->left!=NULL)
        q1.push(f->left);
    if(f->right!=NULL)
        q1.push(f->right);
}
}

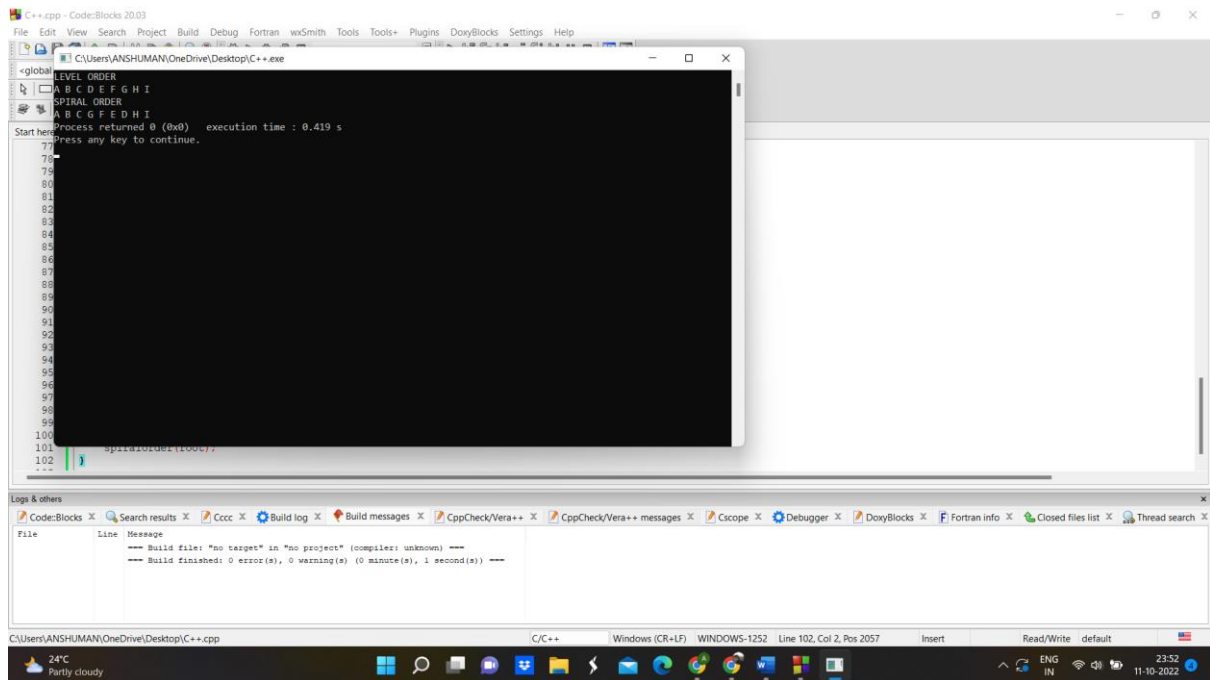
void spiralorder(node *root)
{
    stack<struct node*>s1;
    stack<struct node*>s2;
    s1.push(root);
    while(!s1.empty()||!s2.empty())
    {
        while(!s1.empty())
        {
            node *temp1=s1.top();
            cout<<temp1->data<<" ";
            s1.pop();
            if(temp1->right)
                s2.push(temp1->right);
            if(temp1->left)
                s2.push(temp1->left);
        }
        while(!s2.empty())
        {
            node *temp2=s2.top();
            cout<<temp2->data<<" ";
            s2.pop();
            if(temp2->left)
                s1.push(temp2->left);
            if(temp2->right)
                s1.push(temp2->right);
        }
    }
}

int main()
{
    node *root;
    root=createtree();
    cout<<"LEVEL ORDER"<<endl;
    levelorder(root);
}

```



```
cout<<endl;  
cout<<"SPIRAL ORDER"<<endl;  
spiralorder(root);  
}
```



4)

```
#include <iostream>
```

```
#include<queue>
```

```
using namespace std;
```

```
struct node
```

```
{
```

```
    int data;
```

```
    node *left;
```

```
    node *right;
```

```
};
```

```
int a[]={1,2,3,4,5,6,7,0,8,9,0,0,0,10,11,12,13,12,13,0,14,0,0,0,0,0,15,0,16,17};
```

```
struct node* insertion(int n)
```

```
{
```

```
    node *temp=new node;
```

```
    temp=NULL;
```

```
    if(n>sizeof(a)/sizeof(int))
```

```
    {
```

```
        return temp;
```

```
    }
```

```
    else if(!a[n])
```

```
    {
```

```
        return temp;
```

```
    }
```

```
    else if(a[n])
```

```
    {
```

```
        temp=new node;
```

```
        temp->data=a[n];
```

```
        temp->left=insertion(2*n+1);
```

```

        temp->right=insertion(2*n+2);

    }

}

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

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

void inorder(node *root)
{

```

```

if(root==NULL)
return;
else
{
    inorder(root->left);
    cout<<root->data<<" ";
    inorder(root->right);
}
}

void levelorder(node *root)
{
if (root == NULL) return;
queue < node * > q;
q.push(root);

while (q.empty() == false) {
    node * node = q.front();
    cout << node -> data << " ";
    q.pop();
    if (node -> left != NULL)
        q.push(node -> left);
    if (node -> right != NULL)
        q.push(node -> right);
}
}

```

```

int main() {
node *root;
root=insertion(0);

```

```

preorder(root);

cout<<endl;

inorder(root);

cout<<endl;

postorder(root);

cout<<endl;

levelorder(root);

}

```

The screenshot shows a web browser window with the URL `programiz.com/cpp-programming/online-compiler/`. The page title is "Programiz C++ Online Compiler". A button labeled "Interactive Python Course" is visible in the top right. The main area contains a code editor with a file named `main.cpp`. The code is as follows:

```

1 // C++ program to demonstrate insertion, deletion and traversal in a binary tree
2 #include <iostream>
3 using namespace std;
4
5 struct tnode {
6     int data;
7     tnode *left, *right;
8 };
9
10 tnode* root = NULL;
11
12 void insert(tnode** root, int data) {
13     if (*root == NULL) {
14         *root = new tnode;
15         (*root)->data = data;
16         (*root)->left = NULL;
17         (*root)->right = NULL;
18     } else {
19         if (data < (*root)->data)
20             insert(&(*root)->left, data);
21         else
22             insert(&(*root)->right, data);
23     }
24 }
25
26 void deleteNode(tnode** root, int data) {
27     if (*root == NULL)
28         return;
29     if (data < (*root)->data)
30         deleteNode(&(*root)->left, data);
31     else if (data > (*root)->data)
32         deleteNode(&(*root)->right, data);
33     else {
34         // Node to be deleted
35         if ((*root)->left == NULL & (*root)->right == NULL)
36             *root = NULL;
37         else if ((*root)->left == NULL)
38             *root = (*root)->right;
39         else if ((*root)->right == NULL)
40             *root = (*root)->left;
41         else {
42             // Node has both left and right children
43             tnode* temp = (*root)->left;
44             while (temp->right != NULL)
45                 temp = temp->right;
46             temp->right = (*root)->right;
47             *root = (*root)->left;
48         }
49     }
50 }
51
52 void inorder(tnode* root) {
53     if (root == NULL)
54         return;
55     inorder(root->left);
56     cout << root->data << " ";
57     inorder(root->right);
58 }
59
60 int main() {
61     root = NULL;
62     insert(&root, 10);
63     insert(&root, 20);
64     insert(&root, 30);
65     insert(&root, 40);
66     insert(&root, 50);
67     insert(&root, 40);
68     insert(&root, 35);
69     insert(&root, 25);
70     insert(&root, 20);
71     insert(&root, 40);
72     insert(&root, 18);
73     insert(&root, 19);
74     insert(&root, 22);
75     insert(&root, 27);
76     insert(&root, 30);
77     insert(&root, 27);
78     cout << "INORDER\n";
79     inorder(root);
80     deleteNode(&root, 25);
81     deleteNode(&root, 40);
82     deleteNode(&root, 27);
83     cout << endl;
84     cout << "INORDER AFTER DELETION\n";
85     inorder(root);
86 }

```

The output window shows the following text:

```

/tmp/652wc2UNzs.o
INORDER
10 18 19 20 22 25 27 30 35 40 40 50
INORDER AFTER DELETION
10 18 19 20 22 20 27 30 35 30 40 40 50

```

The Windows taskbar at the bottom shows the date and time as 11-10-2022, 23:46, and the weather as 24°C Partly cloudy.