```
/*

 Beginning with an empty binary tree, construct binary tree by inserting the values in the order given.
 After constructing a binary tree perform following operations on it-
 •Perform in-order, pre-order and post order traversal(Implement both recursive and non-recursive methods)
 •Change a tree so that the roles of the left and right pointers are swapped at every node
 •Find the height of tree
 •Copy this tree to another [operator=]
 •Count number of leaves, number of internal nodes.
 •Erase all nodes in a binary tree.

*/



#include<iostream>
using namespace std;



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

class Stack{
node *s[20]; 29            public:
            int top;
            Stack(){
            top = -1;
            }
            void push(node *);
            node * pop();
```

```
36              bool isempty();
37              };
38
39          void Stack::push(node *t){
40          if(top==19)
41          cout<<"Stack is full!"<<endl;
42        else{
43          top++;
44          s[top]=t;
45          }
46          }
47
48              node * Stack::pop(){
49              node *x;
50              if(top==-1){
51              cout<<"Stack empty!"<<endl;
52              return NULL;
53           }else{
54
x=s[top]; 55
top--;
56                  return x;
57          }
58          }
59
60          bool Stack::isempty(){
61          if(top==-1){
62                  return true; 63          }else{
64                  return false;
65          }
66          }
67
68
69   class BinaryTree{
70
71   72
public:
73        node * root;
74        int lfcount=0,incount=0;
75        BinaryTree();
```

```
76        node * create();
77        void inorder(node * temp);
78        void preorder(node * temp);
79        void postorder(node * temp);
80
81        void allLeafNode(node *temp);
82        void internalNodes(node *temp);
83
84        void nonrecPreorder(node *temp);
85        void nonrecInorder(node *temp);
86        void nonrecPostorder(node *temp);
87
88         node * mirror(node *temp);
89
90         node * copyTree(node *temp);
91
92         void operator = (BinaryTree &);
93
94         void deleteTree(node *temp);
95
96        int heightTree(node *temp);
97        };
98
99    BinaryTree::BinaryTree(){
100
101       root=NULL;
102       }
103
104
105       int BinaryTree::heightTree(node *temp){
106       int leftHeight, rightHeight;
107       if(temp == NULL){
108            return 0; 109        }else{
110               leftHeight = heightTree(temp->left);
111               rightHeight = heightTree(temp->right);
112               if(leftHeight > rightHeight){
113               return leftHeight + 1;
114            }else{
115           return rightHeight + 1;
116           }
```

```
117             }
118             }
119
120
121     void BinaryTree::deleteTree(node *temp){
122
123             if(temp != NULL){
124             deleteTree(temp->left);
125             deleteTree(temp->right);
126             cout<<temp->data<<"  ";
127             delete temp;
128             }
129             }
130
131     void BinaryTree::operator = (BinaryTree &t){
132
133        root = copyTree(t.root);
134        cout<<"Address of initial tree root:
      "<<t.root<<endl;
135         cout<<"Address of copied tree root:
"<<root<<endl;
136     }
137
138     node * BinaryTree::copyTree(node *temp){
139
140             node *t = NULL;
141             if(temp != NULL){
142             t = new node;
143             t->data = temp->data;
144             t->left = copyTree(temp->left);
145             t->right = copyTree(temp->right);
146             }
147         return t;
148     }
149
150     node * BinaryTree::mirror(node *temp){
151
152             node *t = NULL;
153             if(temp != NULL){
154             t = new node;
```

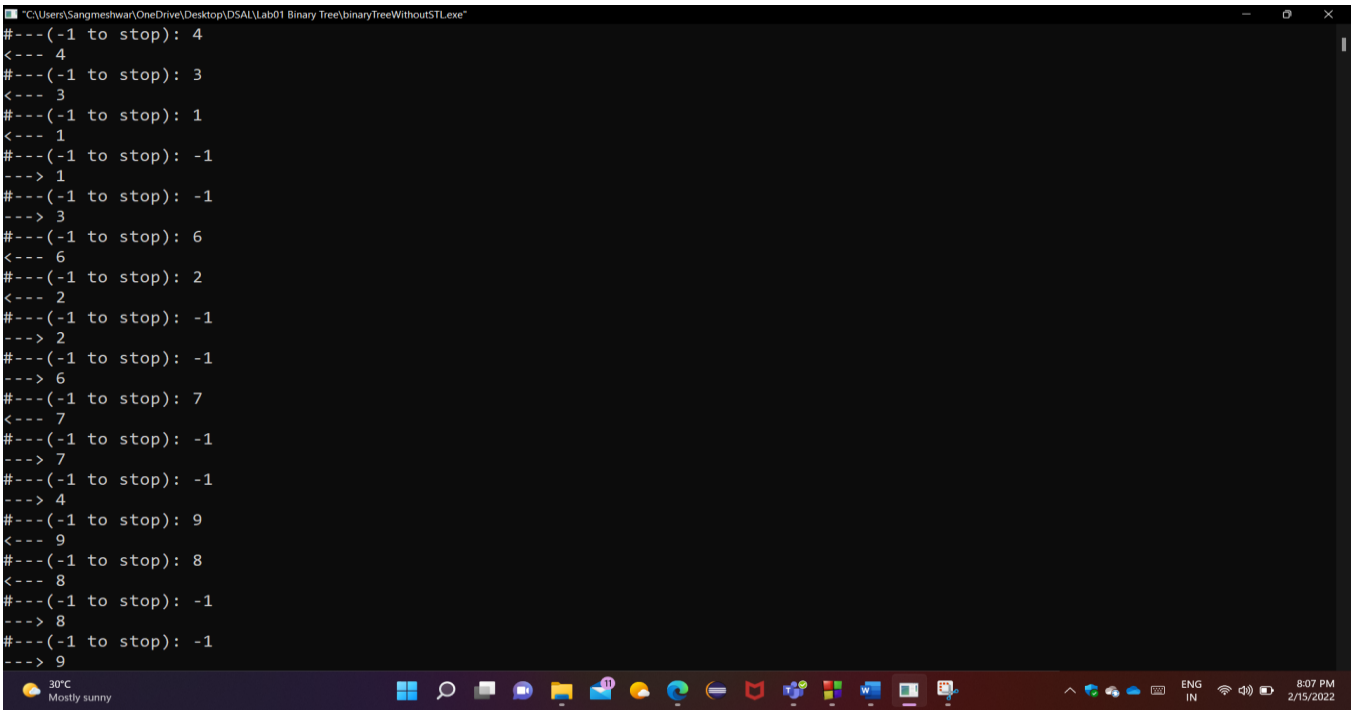
```
155          t->data = temp->data;
156          t->left = mirror(temp->right);
157          t->right = mirror(temp->left);
158          }
159      return t;
160  }
161
162          void BinaryTree::nonrecPostorder(node
             *temp){
163          Stack s1,s2;
164          s1.push(temp);
165          while(!s1.isempty()){
166          temp=s1.pop(); 167
             s2.push(temp);
168                  if(temp->left!=NULL)
169                  s1.push(temp->left); 170
                     if(temp->right!=NULL)
171                   s1.push(temp->right);
172                   }
173                   while(!s2.isempty()){
174                   node* t=s2.pop();
175                   cout<<t->data<<"  ";
176                   }
177                   }
178
179  void BinaryTree::nonrecInorder(node *temp){
180
181              Stack s;
182              while(temp != NULL){
183              s.push(temp);
184              temp = temp->left;
185              }
186              while(!s.isempty()){
187              temp=s.pop();
188              cout<<temp->data<<"  ";
189              temp=temp->right;
190              while(temp!=NULL){
191              s.push(temp);
192              temp=temp->left;
193              }
```

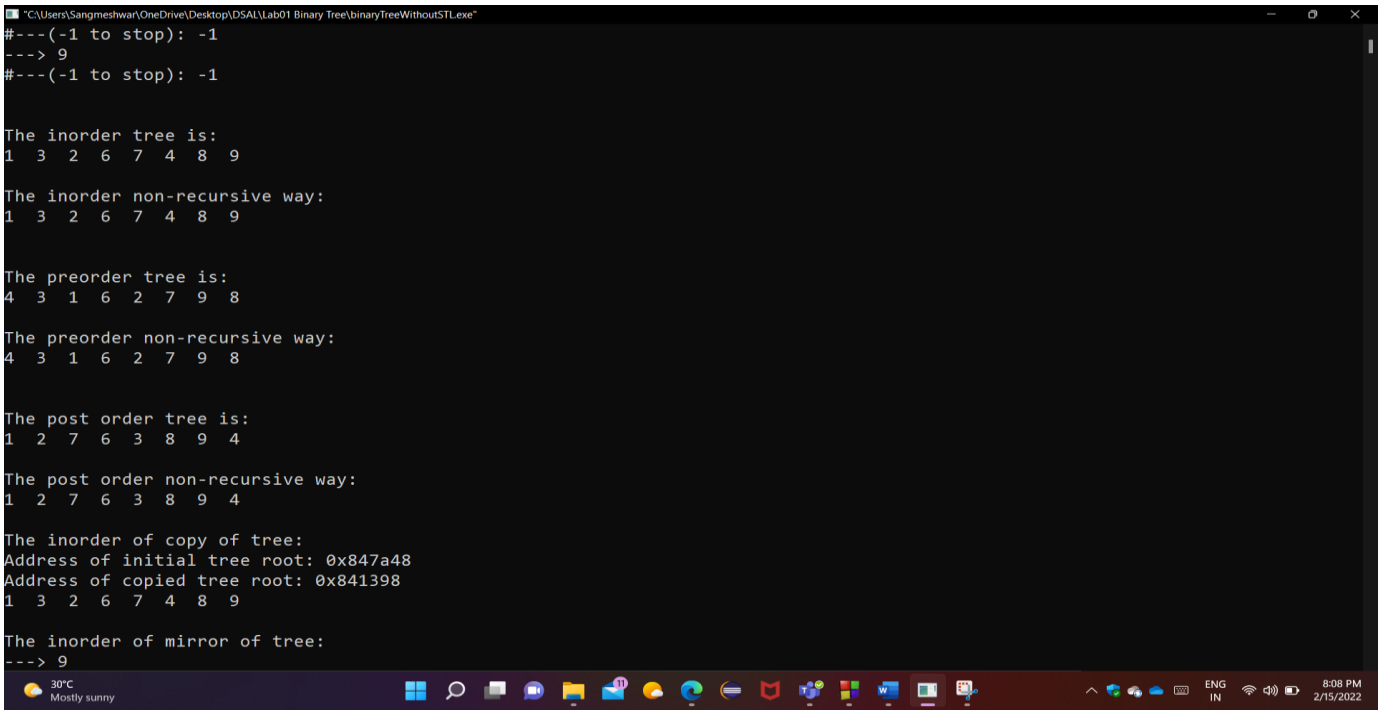
```
194                 }
195                 cout<<endl;
196                 }
197
198     void BinaryTree::nonrecPreorder(node *temp){
199
200                 Stack s;
201                 s.push(temp);
202                 while(!s.isempty()){
203                 temp = s.pop();
204                 cout<<temp->data<<"  ";
205                 if(temp->right!=NULL){
206                 s.push(temp->right);
207                 }
208                 if(temp->left!=NULL){
209                 s.push(temp->left);
210                 }
211                 }
212                 cout<<endl;
213                 }
214
215     void BinaryTree::inorder(node *temp){
216
217                  if(temp != NULL){
218                  inorder(temp->left);
219                  cout<<temp->data<<"  ";
220                  inorder(temp->right);
221                  }
222                  }
223
224                 void BinaryTree::preorder(node *temp){
225                 if(temp != NULL){
226                 cout<<temp->data<<"  ";
227                 preorder(temp->left);
228                 preorder(temp->right);
229                 }
230                 }
231
232
```

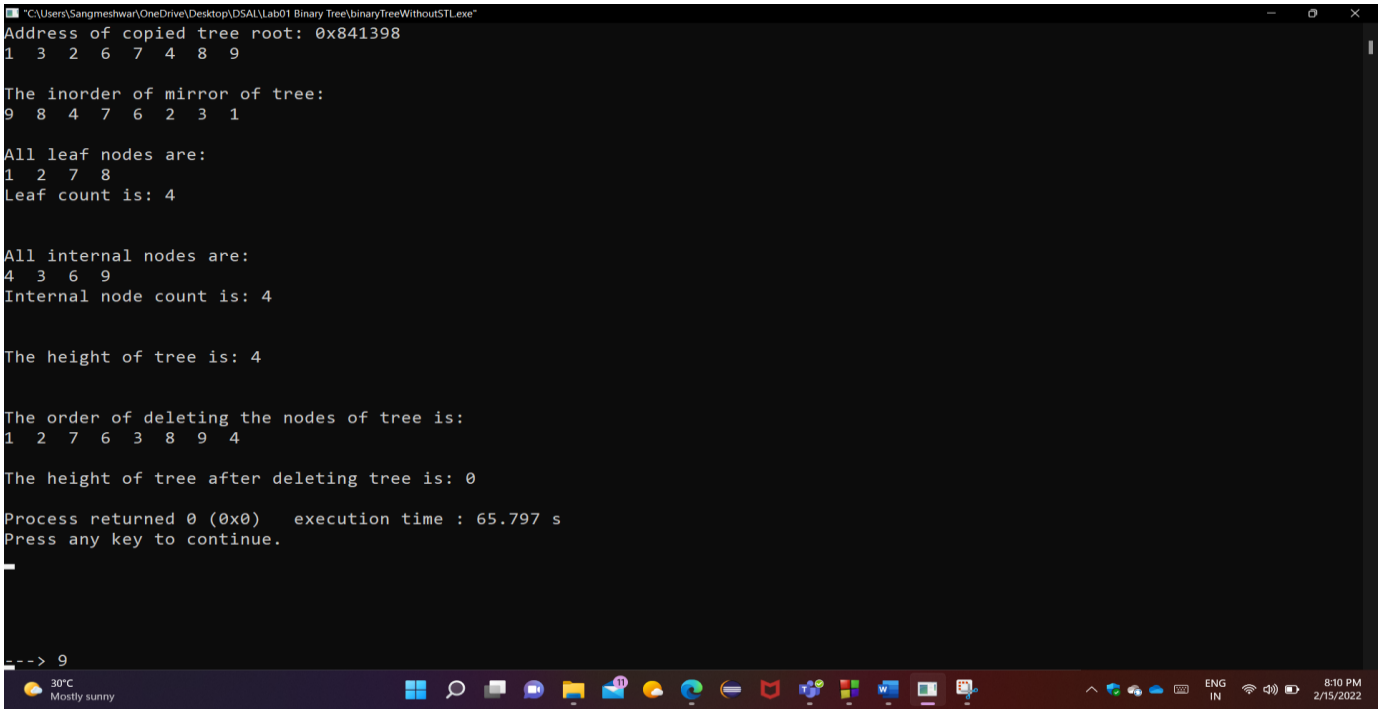
```
233             void BinaryTree::postorder(node
                *temp){
234             if(temp != NULL){
235             postorder(temp->left);
236             postorder(temp->right);
237             cout<<temp->data<<"  ";
238             }
239             }
240
241
242     void BinaryTree::allLeafNode(node *temp){
243
244             if(temp == NULL){
245                 return;
246               }
247               if(temp->left==NULL && temp-
                  >right==NULL){
248               cout<<temp->data<<"  ";
249               lfcount += 1;
250                 return;
251               }
252               if(temp->left != NULL){
253               allLeafNode(temp->left);
254               }
255               if(temp->right != NULL){
256               allLeafNode(temp->right);
257               }
258               }
259
260     void BinaryTree::internalNodes(node *temp){
261
262             if(temp==NULL){
263                 return;
264               }
265               if(temp->left != NULL || temp->right
                  != NULL){
266               cout<<temp->data<<"  ";
267               incount += 1;
268               }
269               if(temp->left != NULL){
```

```
270                 internalNodes(temp->left);
271                 }
272                 if(temp->right != NULL){
273                 internalNodes(temp->right);
274                 }
275                 }
276
277   node * BinaryTree::create(){
278
279           int x;
280           cout<<"#---(-1 to stop): "; //node data
281                 cin>>x;
282                 if(x== -1){
283                 return NULL;
284                 }
285           else{
286                 node *p=new node;
287                 p->data=x;
288                 cout<<"<--- "<<x<<endl; //left
                    child
289                 p->left=create();
290                 cout<<"---> "<<x<<endl; //right
                    child
291                 p->right=create();
292                   return p;
293           }
294           }
295
296   int main(){
297   BinaryTree obj, cobj;
298   node * t;
299   int h;
300   t = obj.create();
301   obj.root = t;
302   cout<<"\n\nThe inorder tree is: "<<endl;
303   obj.inorder(t);
304    cout<<"\n\nThe inorder non-recursive way:
"<<endl;
305   obj.nonrecInorder(t);
306
```

```
307    cout<<"\n\nThe preorder tree is: "<<endl;
308    obj.preorder(t);
309    cout<<"\n\nThe preorder non-recursive way:
       "<<endl; 310    obj.nonrecPreorder(t);
311
312    cout<<"\n\nThe post order tree is: "<<endl;
313    obj.postorder(t);
314     cout<<"\n\nThe post order non-recursive way:
"<<endl;
315     obj.nonrecPostorder(t);
316
317     cout<<"\n\nThe inorder of copy of tree: "<<endl;
318    cobj = obj;
319    obj.inorder(cobj.root);
320
321     cout<<"\n\nThe inorder of mirror of tree:
"<<endl;
322     obj.inorder(obj.mirror(t));
323
324    cout<<"\n\nAll leaf nodes are: "<<endl;
325    obj.allLeafNode(t);
326    cout<<"\nLeaf count is: "<<obj.lfcount<<endl;
327
328    cout<<"\n\nAll internal nodes are: "<<endl;
329    obj.internalNodes(t);
330     cout<<"\nInternal node count is:
"<<obj.incount<<endl;
331
332    h = obj.heightTree(t);
333    cout<<"\n\nThe height of tree is: "<<h<<endl;
334
335     cout<<"\n\nThe order of deleting the nodes of
       tree is: "<<endl;
336    obj.deleteTree(t);
337    t = NULL;
338    h = obj.heightTree(t);
339    cout<<"\n\nThe height of tree after deleting tree
       is: "<<h<<endl;
340     return 0;
341     }
342
```


"C:\Users\Sangmeshwar\OneDrive\Desktop\DSAL\Lab01 Binary Tree\binaryTreeWithoutSTL.exe"
#---(-1 to stop): 4
<--- 4
#---(-1 to stop): 3
<--- 3
#---(-1 to stop): 1
<--- 1
#---(-1 to stop): -1
---> 1
#---(-1 to stop): -1
---> 3
#---(-1 to stop): 6
<--- 6
#---(-1 to stop): 2
<--- 2
#---(-1 to stop): -1
---> 2
#---(-1 to stop): -1
---> 6
#---(-1 to stop): 7
<--- 7
#---(-1 to stop): -1
---> 7
#---(-1 to stop): -1
---> 4
#---(-1 to stop): 9
<--- 9
#---(-1 to stop): 8
<--- 8
#---(-1 to stop): -1
---> 8
#---(-1 to stop): -1
---> 9
30°C
Mostly sunny
ENG
IN
8:07 PM
2/15/2022


"C:\Users\Sangmeshwar\OneDrive\Desktop\DSAL\Lab01 Binary Tree\binaryTreeWithoutSTL.exe"
#---(-1 to stop): -1
---> 9
#---(-1 to stop): -1
The inorder tree is:
1 3 2 6 7 4 8 9
The inorder non-recursive way:
1 3 2 6 7 4 8 9
The preorder tree is:
4 3 1 6 2 7 9 8
The preorder non-recursive way:
4 3 1 6 2 7 9 8
The post order tree is:
1 2 7 6 3 8 9 4
The post order non-recursive way:
1 2 7 6 3 8 9 4
The inorder of copy of tree:
Address of initial tree root: 0x847a48
Address of copied tree root: 0x841398
1 3 2 6 7 4 8 9
The inorder of mirror of tree:
---> 9
30°C
Mostly sunny
ENG
IN
8:08 PM
2/15/2022


"C:\Users\Sangmeshwar\OneDrive\Desktop\DSAL\Lab01 Binary Tree\binaryTreeWithoutSTL.exe"
Address of copied tree root: 0x841398
1 3 2 6 7 4 8 9
The inorder of mirror of tree:
9 8 4 7 6 2 3 1
All leaf nodes are:
1 2 7 8
Leaf count is: 4
All internal nodes are:
4 3 6 9
Internal node count is: 4
The height of tree is: 4
The order of deleting the nodes of tree is:
1 2 7 6 3 8 9 4
The height of tree after deleting tree is: 0
Process returned 0 (0x0) execution time : 65.797 s
Press any key to continue.
--> 9
30°C
Mostly sunny
ENG
IN
8:10 PM
2/15/2022