

```
1  #include <iostream>
2  #include <iomanip>
3  #include <fstream>
4  #include <string>
5
6  using namespace std;
7
8  const int maxstack=40;
9
10 struct treetype {
11     int key;
12     treetype *left, *right;
13 };
14
15 void planttree(treetype *&root) {
16     root = new treetype;
17     root->key=-1;
18     root->left=NULL;
19     root->right=NULL;
20 }
21
22 bool emptytree(treetype *root) {
23     return root->key== -1;
24 }
25
26 void inserttree(treetype *root, int key){
27     treetype *c, *parent, *insert;
28     if(!emptytree(root)) {
29         insert = new treetype;
30         insert->key=key;
31         insert->left=NULL;
32         insert->right=NULL;
33         parent=NULL;
34         c=root;
35         while(c!=NULL) {
36             parent=c;
37             if(insert->key < c->key) c=c->left;
38             else c=c->right;
39         }
40         if(insert->key < parent -> key) parent->left=insert;
41         else parent->right=insert; }
42     else root->key=key;
43 }
44
45 void readinfile(treetype *root){
46     ifstream infile;
47     infile.open("inputfileprogram4.txt");
48     int insertnew;
49     while(!infile.eof()) {
50         infile >> insertnew >> ws;
51         inserttree(root,insertnew); }
52 }
53
54 void insertspaces(ofstream &outf){
```

```

55     outf << endl << endl << endl;
56 }
57
58 void headerinordertraverse(ofstream &outf){
59     outf << endl << setw(12) << setfill(' ') << right << " " << "In-Order Traversal
    of Tree" << endl;
60     outf << setw(49) << setfill('-') << "-" <<endl;
61 }
62
63 void headerpreordertraverse(ofstream &outf){
64     outf << endl << setw(12) << setfill(' ') << right << " " << "Pre-Order Traversal
    of Tree" << endl;
65     outf << setw(49) << setfill('-') << "-" <<endl;
66 }
67
68 void headerpostordertraverse(ofstream &outf){
69     outf << endl << setw(12) << setfill(' ') << right << " " << "Post-Order
    Traversal of Tree" << endl;
70     outf << setw(49) << setfill('-') << "-" <<endl;
71 }
72
73 void inorderinner(treetype *c, ofstream &outf) {
74     if(c->left!=NULL) inorderinner(c->left,outf);
75     outf << c->key<< " ";
76     if(c->right!=NULL)inorderinner(c->right,outf);
77 }
78
79 void inordertraverse(treetype *root, ofstream &outf) {
80     if(!emptytree(root)) {
81         headerinordertraverse(outf);
82         inorderinner(root,outf); }
83     else outf << "Unable to In-Order Traversal because the tree is empty" << endl;
84     insertspaces(outf);
85 }
86
87 void preorderinner(treetype *c, ofstream &outf) {
88     outf << c->key<< " ";
89     if(c->left!=NULL) preorderinner(c->left,outf);
90     if(c->right!=NULL) preorderinner(c->right,outf);
91 }
92
93 void preordertraverse(treetype *root, ofstream &outf) {
94     if(!emptytree(root)) {
95         headerpreordertraverse(outf);
96         preorderinner(root,outf); }
97     else outf << "Unable to Pre-Order Traversal because the tree is empty" << endl;
98     insertspaces(outf);
99 }
100
101 void postorderinner(treetype *c, ofstream &outf) {
102     if(c->left!=NULL) postorderinner(c->left,outf);
103     if(c->right!=NULL)postorderinner(c->right,outf);
104     outf << c->key<< " ";
105 }

```

```

106
107 void postordertraverse(treetype *root, ofstream &outf) {
108     if(!emptytree(root)) {
109         headerpostordertraverse(outf);
110         postorderinner(root,outf);    }
111     else outf << "Unable to Post-Order Traverse because the tree is empty" << endl;
112     insertspaces(outf);
113 }
114
115 void deletealeaf(treetype *parent, treetype *current) {
116     if (current->key < parent->key) parent->left=NULL;
117     else parent->right=NULL;
118     delete current;
119 }
120
121 void deletesinglechild(treetype *parent, treetype *current){
122     treetype *child;
123     if (current->left != NULL) child = current->left;
124     else child=current->right;
125     if(current->key < parent->key) parent->left=child;
126     else parent ->right=child;
127     delete current;
128 }
129
130 void deletedoublechild(treetype *current){
131     treetype *replace;
132     treetype *parentofreplace;
133     replace=current->left;
134     parentofreplace=current;
135     while (replace->right!=NULL) {
136         parentofreplace=replace;
137         replace=replace->right; }
138     current->key=replace->key;
139     if(replace->left==NULL) deletealeaf(parentofreplace,replace);
140     else deletesinglechild(parentofreplace,replace);
141 }
142
143
144 void deletefromtree(treetype *root, int key, ofstream &outf) {
145     treetype *current;
146     treetype *parent;
147     parent = NULL;
148     current=root;
149     while(current!=NULL && key!=current->key){
150         parent=current;
151         if(key<current->key) current=current->left;
152         else current=current->right;
153     }
154     if(current->key==key && current!=NULL){
155         if(current->left==NULL && current->right==NULL) deletealeaf(parent,current);
156         else if(current->left!=NULL&&current->right!=NULL) deletedoublechild(current);
157         else deletesinglechild(parent,current);
158     }
159     else outf <<"Key, " << key << " was not found." << endl;

```

```

160     }
161
162     void headeriterativeinordertraverse(ofstream &outf){
163         outf << endl << setw(6) << setfill(' ') << right << " " << "Iterative In-Order Traversal of Tree" << endl;
164         outf << setw(49) << setfill('-') << "-" <<endl;
165     }
166
167     void headeriterativepreordertraverse(ofstream &outf){
168         outf << endl << setw(6) << setfill(' ') << right << " " << "Iterative Pre-Order Traversal of Tree" << endl;
169         outf << setw(49) << setfill('-') << "-" <<endl;
170     }
171
172     bool EmptyStack(int top) {
173         return top < 0;
174     }
175
176     bool FullStack(int top) {
177         return top >= maxstack-1;
178     }
179
180     void push(treetype* Stack[], int &top, treetype* data) {
181         if(!FullStack(top)) {
182             top++;
183             Stack[top]=data; }
184     }
185
186     treetype* pop(treetype* Stack[], int &top) {
187         treetype* temp;
188         if (!EmptyStack(top)) {
189             temp = Stack[top];
190             top--;
191         }
192         return temp;
193     }
194
195     void iterativepreordertraversal(ofstream &outf,treetype* root, int top){
196         headeriterativepreordertraverse(outf);
197         treetype* Stack[maxstack];
198         treetype* c;
199         if(!emptytree(root)) {
200             push(Stack,top,NULL);
201             c=root;
202             while (c!=NULL) {
203                 outf<<c->key <<" ";
204                 if(c->right!=NULL) push(Stack,top,c->right);
205                 if(c->left!=NULL) c=c->left;
206                 else c=pop(Stack,top);
207             }
208         }
209         else outf << "Empty Tree" << endl;
210         insertspaces(outf);
211     }

```

```
212
213 void iterativeinordertraversal(ofstream &outf, treetype* root, int top) {
214     headeriterativeinordertraverse(outf);
215     treetype* Stack[maxstack];
216     treetype* c;
217     bool done;
218     if(!emptytree(root)) {
219         push(Stack, top, NULL);
220         c=root;
221         while(c!=NULL) {
222             while(c->left!=NULL) {
223                 push(Stack,top,c);
224                 c=c->left; }
225             done = false;
226
227             while (!done) {
228                 outf << c->key << " ";
229                 if(c->right!=NULL) {
230                     c=c->right;
231                     done=true; }
232                 else {
233                     c=pop(Stack,top);
234                     if(c==NULL) done=true; }
235             } } }
236     else outf << "Empty Tree" << endl;
237     insertspaces(outf);
238 }
239
240 int main() {
241     treetype *root;
242     int top=-1;
243     ofstream outfile;
244     outfile.open("outputfileprogram4.txt");
245     planttree(root);
246     readinfile(root);
247     inordertraverse(root,outfile);
248     deletefromtree(root,71,outfile);
249     postordertraverse(root,outfile);
250     deletefromtree(root,38,outfile);
251     preordertraverse(root,outfile);
252     iterativepreordertraversal(outfile,root,top);
253     iterativeinordertraversal(outfile,root,top);
254 }
255
```