```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>

using namespace std;

const int maxstack=40;

struct treetype {
    int key;
    treetype *left, *right;
};

void planttree(treetype *&root) {
    root = new treetype;
    root->key=-1;
    root->left=NULL;
    root->right=NULL;
}

bool emptytree(treetype *root) {
    return root->key==-1;
}

void inserttree(treetype *root, int key){
    treetype *c, *parent, *insert;
    if(!emptytree(root)){
        insert = new treetype;
        insert->key=key;
        insert->left=NULL;
        insert->right=NULL;
        parent=NULL;
        c=root;
        while(c!=NULL) {
            parent=c;
            if(insert->key < c->key) c=c->left;
            else c=c->right;
        }
        if(insert->key < parent -> key) parent->left=insert;
        else parent->right=insert; }
    else root->key=key;
}

void readinfile(treetype *root){
ifstream infile;
infile.open("inputfileprogram4.txt");
int insertnew;
while(!infile.eof()) {
    infile >> insertnew >> ws;
    inserttree(root,insertnew); }
}

void insertspaces(ofstream &outf){
    outf << endl << endl << endl;
}

void headerinordertraverse(ofstream &outf){
    outf << endl << setw(12) << setfill(' ') << right << " " << "In-Order Traversal of Tree" << endl;
    outf << setw(49) << setfill('-') << "-" <<endl;
}

void headerpreordertraverse(ofstream &outf){
    outf << endl << setw(12) << setfill(' ') << right << " " << "Pre-Order Traversal of Tree" << endl;
    outf << setw(49) << setfill('-') << "-" <<endl;
}

void headerpostordertraverse(ofstream &outf){
    outf << endl << setw(12) << setfill(' ') << right << " " << "Post-Order Traversal of Tree" << endl;
    outf << setw(49) << setfill('-') << "-" <<endl;
}

void inorderinner(treetype *c, ofstream &outf) {
    if(c->left!=NULL) inorderinner(c->left,outf);
    outf << c->key<< " ";
    if(c->right!=NULL)inorderinner(c->right,outf);
}

void inordertraverse(treetype *root, ofstream &outf) {
    if(!emptytree(root)){
        headerinordertraverse(outf);
        inorderinner(root,outf); }
    else outf << "Unable to In-Order Traverse because the tree is empty" << endl;
    insertspaces(outf);
}

void preorderinner(treetype *c, ofstream &outf) {
    outf << c->key<< " ";
    if(c->left!=NULL) preorderinner(c->left,outf);
    if(c->right!=NULL) preorderinner(c->right,outf);
}

void preordertraverse(treetype *root, ofstream &outf) {
    if(!emptytree(root)) {
        headerpreordertraverse(outf);
        preorderinner(root,outf);  }
    else outf << "Unable to Pre-Order Traverse because the tree is empty" << endl;
    insertspaces(outf);
}

void postorderinner(treetype *c, ofstream &outf) {
    if(c->left!=NULL) postorderinner(c->left,outf);
    if(c->right!=NULL)postorderinner(c->right,outf);
    outf << c->key<< " ";
}
```

```cpp

void postordertraverse(treetype *root, ofstream &outf) {
    if(!emptytree(root)) {
        headerpostordertraverse(outf);
        postorderinner(root,outf);  }
    else outf << "Unable to Post-Order Traverse because the tree is empty" << endl;
    insertspaces(outf);
}

void deletealeaf(treetype *parent, treetype *current) {
    if (current->key < parent->key) parent->left=NULL;
    else parent->right=NULL;
    delete current;
}

void deletesinglechild(treetype *parent, treetype *current){
    treetype *child;
    if (current->left != NULL) child = current->left;
    else child=current->right;
    if(current->key < parent->key) parent->left=child;
    else parent ->right=child;
    delete current;
}

void deletedoublechild(treetype *current){
    treetype *replace;
    treetype *parentofreplace;
    replace=current->left;
    parentofreplace=current;
    while (replace->right!=NULL) {
        parentofreplace=replace;
        replace=replace->right; }
    current->key=replace->key;
    if(replace->left==NULL) deletealeaf(parentofreplace,replace);
    else deletesinglechild(parentofreplace,replace);
}


void deletefromtree(treetype *root, int key, ofstream &outf) {
    treetype *current;
    treetype *parent;
    parent = NULL;
    current=root;
    while(current!=NULL && key!=current->key){
        parent=current;
        if(key<current->key) current=current->left;
        else current=current->right;
    }
    if(current->key==key && current!=NULL){
        if(current->left==NULL && current->right==NULL) deletealeaf(parent,current);
        else if(current->left!=NULL&&current->right!=NULL) deletedoublechild(current);
        else deletesinglechild(parent,current);
    }
    else outf <<"Key, " << key << " was not found." << endl;
}

void headeriterativeinordertraverse(ofstream &outf){
    outf << endl << setw(6) << setfill(' ') << right << " " << "Iterative In-Order Traversal of Tree" << endl;
    outf << setw(49) << setfill('-') << "-" <<endl;
}

void headeriterativepreordertraverse(ofstream &outf){
    outf << endl << setw(6) << setfill(' ') << right << " " << "Iterative Pre-Order Traversal of Tree" << endl;
    outf << setw(49) << setfill('-') << "-" <<endl;
}

bool EmptyStack(int top) {
    return top < 0;
}

bool FullStack(int top) {
    return top >= maxstack-1;
}

void push(treetype* Stack[], int &top, treetype* data) {
    if(!FullStack(top)) {
        top++;
        Stack[top]=data; }
}

treetype* pop(treetype* Stack[], int &top) {
    treetype* temp;
    if (!EmptyStack(top)) {
        temp = Stack[top];
        top--;
    }
    return temp;
}

void iterativepreordertraversal(ofstream &outf,treetype* root, int top){
    headeriterativepreordertraverse(outf);
    treetype* Stack[maxstack];
    treetype* c;
    if(!emptytree(root)){
        push(Stack,top,NULL);
        c=root;
        while(c!=NULL){
            outf<<c->key <<" ";
            if(c->right!=NULL) push(Stack,top,c->right);
            if(c->left!=NULL) c=c->left;
            else c=pop(Stack,top);
        }
    }
    else outf << "Empty Tree" << endl;
    insertspaces(outf);
}
```

```

void iterativeinordertraversal(ofstream &outf, treetype* root, int top) {
    headeriterativeinordertraverse(outf);
    treetype* Stack[maxstack];
    treetype* c;
    bool done;
    if(!emptytree(root)){
        push(Stack, top, NULL);
        c=root;
        while(c!=NULL){
            while(c->left!=NULL){
                push(Stack,top,c);
                c=c->left; }
            done = false;

        while (!done){
            outf << c->key << " ";
            if(c->right!=NULL){
                c=c->right;
                done=true; }
            else {
                c=pop(Stack,top);
                if(c==NULL) done=true; }
        } } }
    else outf << "Empty Tree" << endl;
    insertspaces(outf);
}

int main() {
    treetype *root;
    int top=-1;
    ofstream outfile;
    outfile.open("outputfileprogram4.txt");
    planttree(root);
    readinfile(root);
    inordertraverse(root,outfile);
    deletefromtree(root,71,outfile);
    postordertraverse(root,outfile);
    deletefromtree(root,38,outfile);
    preordertraverse(root,outfile);
    iterativepreordertraversal(outfile,root,top);
    iterativeinordertraversal(outfile,root,top);
}

```