```cpp
///Marc Pfeiffer
///Assignment 4 - Trees

#include <iostream>
#include <fstream>

using namespace std;

ifstream dataIn("infile.txt");
ofstream dataOut("outfile.txt");


class Node{

public:
    int num;
    Node* left;
    Node* right;
    Node* father;
};

Node* makeTree(int );
void setLeft(Node* , int );
void setRight(Node* , int );
void inTrav(Node*  );
void preTrav(Node* );
void postTrav(Node*  );
int countTrav(Node* );
void children(Node* );
void insrt(Node *, int );
int findSmallest(Node *);
void removeNode(Node *, int );
void removeRootNode(Node* );
void removeMatch(Node*, Node*, bool);

///Makes tree
Node* makeTree(int x){

    Node* p;
    p = new Node;
    p->num = x;
    p->right=NULL;
    p->left =NULL;
    p->father = NULL;

    return p;
}

///Sets node to left Son
void setLeft(Node* p, int x){

    Node* Q;
    Q = new Node;
    Q->num = x;
    Q->right = NULL;
    Q->left = NULL;
    Q->father = p;
    p->left = Q;

}

///Sets node to right Son
void setRight(Node* p, int x){

    Node* Q;
    Q = new Node;
```

```
    Q->num = x;
    Q->right = NULL;
    Q->left = NULL;
    Q->father = p;
    p->right = Q;


}

///prints out tree in-order
 void inTrav(Node* p ){

    if(p != NULL){

        inTrav(p->left);
        dataOut<<p->num<<" ";
        inTrav(p->right);

    }
 }

///prints out tree Pre-Order
void preTrav(Node* p ){

    if(p!= NULL){

        dataOut<<p->num<<" ";
        preTrav(p->left);
        preTrav(p->right);

    }
 }

///Prints out tree post-order
 void postTrav(Node* p ){

    if(p!= NULL){

        postTrav(p->left);
        postTrav(p->right);
        dataOut<<p->num<<" ";

    }
 }

///counts Number of Nodes
 int countTrav(Node* p){

    int counter=0;

    if(p){

        counter += countTrav(p->left) ;
        counter +=countTrav(p->right) ;

        counter++;

    }

    return counter;
 }

///prints out how many children each node has
 void children(Node* p){

    if(p!= NULL){
```

```
133         children(p->left);
134     if(p->left != NULL && p->right != NULL){
135         dataOut<<"Node "<<p->num<<" has 2 childern"<<endl;
136     }
137     else if((p->left == NULL && p->right != NULL)||(p->left != NULL && p->right ==
NULL) ){
138         dataOut<<"Node "<<p->num<<" has 1 child"<<endl;
139 
140     }
141     else{
142         dataOut<<"Node "<<p->num<<" has 0 children"<<endl;
143     }
144         children(p->right);
145     }
146  }
147 ///insert function
148 void insrt(Node *Tree, int x){
149     Node *Q, *P;
150     Q = P = Tree;
151     if(Tree == NULL){
152         Tree = makeTree(x);
153         return;
154     }
155 
156     while((x!= P->num) &&( Q!= NULL)){
157 
158         P=Q;
159         if(x< P->num){
160             Q= Q->left;
161 
162         }
163         else{
164             Q=Q->right;
165         }
166     }
167          if(P->num==x){
168 
169            dataOut<<"Duplicate";
170 
171         }
172         else  if( x  < (P->num)){
173             setLeft(P,x);
174         }
175         else{
176             setRight(P,x);
177     }
178 
179 
180 
181 }
182 
183 /// find the smallest Node
184 int findSmallest(Node *Tree){
185 
186     if(Tree == NULL){
187 
188         dataOut<<"tree is empty"<<endl;
189         return -9999;
190 
191     }
192     else{
193 
194         if(Tree->left!=NULL){
195 
196             return findSmallest(Tree->left);
197         }
```

```
198         else{
199 
200             return Tree->num;
201         }
202     }
203 }
204 
205 ///remove node if its the root
206 void removeRootNode(Node* root){
207 
208     if(root!=NULL){
209 
210         Node* delptr = root;
211         int value = root->num;
212         int smallest;
213 
214         if(root->left == NULL && root->right == NULL){
215 
216             root =NULL;
217             delete delptr;
218         }
219         else if(root->left == NULL && root->right != NULL){
220 
221             root = root->right;
222             delptr->right = NULL;
223             root->father = NULL;
224             delete delptr;
225 
226         }
227         else if(root->left != NULL && root->right == NULL){
228 
229             root = root->left;
230             delptr->left = NULL;
231             root->father = NULL;
232             delete delptr;
233 
234         }
235         else{
236 
237             smallest = findSmallest(root->right);
238             removeNode(root, smallest);
239             root->num = smallest;
240         }
241 
242 
243     }
244     else{
245 
246         dataOut<<"Cannot remove, Tree is already empty\n";
247     }
248 
249 }
250 
251 ///remove node decides which remove function to use
252 void removeNode(Node *root, int value){
253 
254    if(root!=NULL){
255 
256         if(root->num == value){
257 
258             removeRootNode(root);
259         }
260         else{
261 
262             if(value < root->num && root->left != NULL){
263 
```

```
                if(root->left->num == value){

                    removeMatch(root, root->left, true);
                }
                else{
                    removeNode(root->left, value);
                }
            }
            else if( value> root->num && root->right != NULL){

                if(root->right->num == value){

                    removeMatch(root, root->right , false);

                }
                else {

                    removeNode(root->right, value);
                }
            }
            else{
                dataOut<<value<<" doesnt exsist withing Tree\n";
            }
        }

    }
    else{

     dataOut<<"The tree is Empty\n";
    }
}

///remove a non-root
void removeMatch(Node* root, Node* match, bool left){

    if(root != NULL){

        Node* delptr;
        int matchValue =match->num;

        if(match->left == NULL && match->right == NULL){

            delptr = match;
            if(left == true){
                root->left = NULL;
            }
            else{
                root->right = NULL;
            }
            delete delptr;

        }
        else if(match->left == NULL && match->right != NULL){


            if(left == true){
                root->left = match->right;
            }
            else{
                root->right = match->right;
            }
            match->right = NULL;
            match->father =NULL;
            delptr= match;
            delete delptr;

```

```
330         }
331            else if(match->left != NULL && match->right == NULL){
332
333
334             if(left == true){
335                 root->left = match->left;
336             }
337             else{
338                 root->right = match->left;
339             }
340             match->left = NULL;
341             match->father = NULL;
342             delptr= match;
343             delete delptr;
344
345         }
346         else{
347             int smallest = findSmallest(match->right);
348             removeNode(match, smallest);
349             match->num = smallest;
350         }
351
352
353     }
354
355     else{
356         dataOut<<"cannot remove, no Match";
357     }
358
359
360 }
361
362
363 int main()
364 {
365     int value;
366
367
368 while(!dataIn.eof()){
369
370     Node *Tree = NULL;
371     dataIn>>value;
372     dataOut<<"\t\t\t\tNEW SET OF DATA\n\n\n\n\n";
373
374     while(value != -999){
375
376         if(Tree == NULL){
377             Tree = makeTree(value);
378         }
379         else{
380             insrt(Tree, value);
381         }
382         dataIn>>value;
383     }
384
385     dataOut<<"\nINORDER   ::  ";inTrav(Tree);dataOut<<endl<<endl;
386     dataOut<<"PREORDER  ::  ";preTrav(Tree); dataOut<<endl<<endl;
387     dataOut<<"POSTORDER :: ";postTrav(Tree);dataOut<<endl<<endl;
388
389
390     int amount = countTrav(Tree);
391
392     dataOut<<"There are "<<amount<<" nodes in this tree"<<endl<<endl;
393
394     children(Tree);
395
```

```
396       string what;
397       dataIn>>what;
398
399       while(what != "new_set"){
400
401           dataIn>>value;
402           if(what == "insert"){
403
404               insrt(Tree, value);
405
406           }
407           else if(what == "delete"){
408
409               removeNode(Tree, value);
410
411           }
412
413           dataIn>>what;
414
415       }
416
417       dataOut<<"\nINORDER   ::  ";inTrav(Tree);dataOut<<endl<<endl;
418       dataOut<<"PREORDER  ::  ";preTrav(Tree); dataOut<<endl<<endl;
419       dataOut<<"POSTORDER :: ";postTrav(Tree);dataOut<<endl<<endl;
420       amount = countTrav(Tree);
421
422       dataOut<<"There are "<<amount<<" nodes in this tree"<<endl<<endl;
423
424       children(Tree);
425
426       delete Tree;
427       dataOut<<"\n\n\n\n\n";
428       }
429
430
431
432
433       return 0;
434   }
435
```