```
//Uchenna Onwudiwe
//CSC 212 - Ali Sabbir
//

#include<iostream>
#include<cstdlib>
#include<ctime>
#include<math.h>

using namespace std;

void is_bst(struct TreeNode* tree, int flag);

struct datatype{
        char * name;
        int id;
        char gender;
        char * major;
        float gpa;};

datatype max(datatype, datatype);
int max(int, int);

struct TreeNode{
        datatype item;
        TreeNode *left;
        TreeNode *right;
        };

class TreeType{
        public:
                        TreeType();
                        TreeType(TreeType &orig); //copy constructor
                        TreeType& operator=(TreeType& rhs);
                        void InsertItem(datatype item);
                        void Print_Tree();
                        int TreeHeight();
                        void DestroyTree();
                        void DeleteItem(char * key);
                        int Size();

        private:
                        TreeNode* root;
                        int TreeSize(TreeNode* root);
                        void Insert(TreeNode* &root, datatype item);
                        void Print(TreeNode* cur);
                        int Height(TreeNode* root);
                        void Delete(TreeNode* &tree, char * key);
```

```
                        void DeleteNode(TreeNode* &tree);
                        void Destroy(TreeNode* root);
                        void GetPredecessor(TreeNode* tree, datatype &data);
                        void Copy(TreeNode* orig_ptr, TreeNode* &copy_ptr);

        };

TreeType::TreeType(){
        root = NULL;
}

TreeType::TreeType(TreeType &orig){
        Copy(orig.root, root);
}

TreeType& TreeType::operator =(TreeType& rhs){
        if(this==&rhs) return *this;
        else{
                Destroy(root);
                Copy(rhs.root, root);
        }
        return *this;
}

int TreeType::Size(){
 return TreeSize(root);}

int TreeType::TreeSize(TreeNode* root){
        if (root==NULL)return 0;
        else return 1 + TreeSize(root_>left) + TreeSize(root_>right);}

void TreeType::InsertItem(datatype item){
        Insert(root, item);
}

void TreeType::Insert(TreeNode* &tree, datatype item){
        if(tree==NULL){
                tree=new TreeNode;
                tree_>right=NULL;
                tree_>left=NULL;
                strcpy(tree_>item.name,item.name);
                tree_>item.id=item.id;
                strcpy(tree_>item.major, item.major);
                tree_>item.gpa=item.gpa;
        }
        else if (strcmp(item.name,tree_>item.name)<0) Insert(tree_>left,item);
        else Insert(tree_>right, item);
}
```

```
int TreeType::TreeHeight(){
return Height(root);}

int TreeType::Height(TreeNode* root){
if (root==NULL) return 0;
else return (1+ max(Height(root_>left), Height(root_>right)));
}


void TreeType::DestroyTree(){
        Destroy(root);
        root=NULL;
}

void TreeType::Destroy(TreeNode* tree){
        if(tree !=NULL){
                Destroy(tree_>left);
                Destroy(tree_>right);
                delete tree;
        }
}

void TreeType::Print(TreeNode* cur){
        if(cur !=NULL){
                Print(cur_>left);
                cout<<"NAME="<<cur_>item.name<<",SSN="<<cur_>item.id<<",
GENDER="<<cur_>item.gender<<cur_>item.major<<"GPA="<<cur_>item.gpa<<endl;
                Print(cur_>right);

        }
}

void TreeType::Print_Tree(){
        Print(root);
}

void TreeType::DeleteItem(char * key){
        Delete(root, key);
}

void TreeType::Delete(TreeNode* &tree, char * key){
        if(tree==NULL){
                cout<<"Student record not found\n";
                return;
        }
        else if (strcmp(key,tree_>item.name)<0) Delete(tree_>left, key);
```

```
        else if (strcmp(key,tree_->item.name)>0) Delete(tree_->right, key);
        else DeleteNode(tree);
}

void TreeType::DeleteNode(TreeNode* &tree){
        datatype data;
        TreeNode* temp;

        temp=tree;
        if(tree_->left==NULL){
                tree=tree_->right;
                delete temp;
        }
        else{
                GetPredecessor(tree_->left, data);
                strcpy(tree_->item.name,data.name);
                tree_->item.id=data.id;
                data.gender=tree_->item.gender;
                tree_->item.gpa=data.gpa;
                Delete(tree_->left, data.name);
        }
}

void TreeType::GetPredecessor(TreeNode* tree, datatype& data){
        while(tree_->right !=NULL)tree=tree_->right;
        strcpy(data.name,tree_->item.name);
        data.id=tree_->item.id;
        strcpy(data.major, tree_->item.major);
        data.gpa=tree_->item.gpa;
}

void TreeType::Copy(TreeNode* orig_ptr, TreeNode* &copy_ptr){
        if(orig_ptr==NULL) copy_ptr=NULL;
        else{
                copy_ptr=new TreeNode;
                copy_ptr_>item=orig_ptr_>item;
                strcpy(copy_ptr_>item.name,orig_ptr_>item.name);
                copy_ptr_>item.id=orig_ptr_>item.id;
                strcpy(copy_ptr_>item.major,orig_ptr_>item.major);
                copy_ptr_>item.gpa=orig_ptr_>item.gpa;

                Copy(orig_ptr_>left, copy_ptr_>left);
                Copy(orig_ptr_>right, copy_ptr_>right);
        }
}

datatype max(datatype x, datatype y){
        if (strcmp(x.name,y.name)>0)
```

```
        return x;
        else return y;
}

int max(int x, int y){return x>y?x:y;}

void is_bst(TreeNode* tree, int flag){
        if(tree==NULL) flag=1;
        else if (tree_>left == NULL && tree_>right==NULL)
                flag=1;
        else if(tree_>left==NULL){
                if(strcmp(tree_>item.name,tree_>right_>item.name) < 0)
                        is_bst(tree_>right, flag);
                else flag=0;
        }
        else if(tree_>right==NULL){
                if(strcmp(tree_>item.name,tree_>right_>item.name) > 0)
                        is_bst(tree_>right, flag);
                else flag=0;
        }
        else if ((strcmp(tree_>right_>item.name,tree_>item.name)>0) &&
(strcmp(tree_>left_>item.name,tree_>item.name)<0)){
                is_bst(tree_>left, flag);
                if(flag!=0)is_bst(tree_>right, flag);
        }
        else flag=0;
}

void main(){

        datatype records;
        TreeType database;
        char * a="abcdefghijklmnopqrstuvwxyz ";
        int counter=0;
        int gen;
        int course;
        records.name="";
        char * b;

        while (counter<=29){
                int i=rand()%26;
                for (int j=0; j<=i; j++)
                        records.name[i]=a[i];

                gen=rand()%2;
                if(gen==0) records.gender='M';
                else records.gender='F';
```

```
        course=rand()%6+1;
        switch (course){
        case 1:
                records.major="CS  ";
                        break;
        case 2:
                records.major="MATH";
                break;
        case 3:
                records.major="ENG ";
                        break;
        case 4:
                records.major="ECON";
                break;
        case 5:
                records.major="PHYS";
                break;
        case 6:
                records.major="CHEM";
                break;
        }

        records.gpa=rand()%4;
        records.id=rand()%1000 + 678;

        database.InsertItem(records);


};

char command='t';

while (command!='q' ||command!='Q'){
        cout<<"\nPlease enter a command key\n"<<
                "\"I\"= insert new student record\n"<<
                "\"D\"= delete existing student record\n"<<
                "\"S\"= retrieve existing student record\n"<<
                "\"P\"= print entire database_:";
        cin>>command;

        if (command=='I' ||command=='i'){
                cout<<"\nenter student name:";
                cin.getline(records.name,25);
                cout<<"\nenter student id:";
                        cin>>records.id;
                while(records.id<0){
                        cout<<"\nenter valid student id:";
                                cin>>records.id;}
```

```
                cout<<"\nenter gender(M or F):";
                cin>>records.gender;
                while(records.gender!='m'
||records.gender!='M'||records.gender!='f'||records.gender!='F'){
                        cout<<"\ninvalid gender! enter gender (m or f)";
                                cin>>records.gender;}
                cout<<"\nenter student major(1=CS, 2=MATH, 3=ENG, 4=ECON,
5=PHYS, 6=CHEM)";
                cin>>course;
                while (course<1 ||course>6){
                        cout<<"\nenter a valid course number!";
                        cin>>course;}
                switch (course){
                case 1:
                        records.major="CS  ";
                        break;
                case 2:
                        records.major="MATH";
                        break;
                case 3:
                        records.major="ENG ";
                        break;
                case 4:
                        records.major="ECON";
                        break;
                case 5:
                        records.major="PHYS";
                        break;
                case 6:
                        records.major="CHEM";
                        break;
        }
                database.InsertItem(records);


        }


                else if (command=='d' ||command=='D'){
                        cout<<"\n enter student name to be deleted:";
                cin.getline(b,25);
                database.DeleteItem(b);
                continue;
                }

                else if (command=='s'||command=='S'){
                        cout<<"\nenter student name to be retrieved";
                        cin.getline(b,25);
```

```
                }
                else if (command=='p'|| command=='P'){
                cout<<"\n";
                database.Print_Tree();
                cout<<"\n tree height is:"<<database.TreeHeight()<<
                        "\n log(1,n) is: "<<ceil(log(database.Size()));}

                else continue;



        }
}
```