/\*This is a simple program to give the student experience writing code

for binary trees. This is a CLASS implementation of the BST ADT.

The student should study, comment, correct errors, compile,

implement/un-implemented/undeveloped functions, and modify code to make

it more efficient when ever necessary. The student should be able to

discuss the advantages and disadvantages of using such an

implementation.\*/

#include <iostream>

#include <string>

using namespace std;

class treenode{

public:

string info;

treenode \*lchild, \*rchild;

};

class BST{

public:

BST(){root=0;};

~BST();

bool empty(){return (root==0);};

void insert(string item);

void insert\_aux(treenode \* &, string item);

void insert\_Iterate(string item);

void del(string item);

void del\_aux(treenode \* & loc\_ptr, string item);

treenode \* search\_tree\_aux(treenode \*,string item);

treenode \* search\_tree(string item);

treenode \* search\_tree\_Iterate(string item);

treenode \* inorder\_succ(treenode \*);

treenode \* parent();

void print\_tree();

void print\_tree\_aux(treenode \*);

int count\_node();

int count\_node(treenode \*);

int count\_node(treenode \*, string r1, string r2);

int count\_node(string r1, string r2);

private:

treenode \*root;};

void BST::insert(string item){

insert\_aux(root,item);}

void BST::insert\_aux(treenode \* & loc\_ptr,string item){

if (loc\_ptr==0){

loc\_ptr = new treenode;

loc\_ptr->lchild=loc\_ptr->rchild=0;

loc\_ptr->info=item;

}

else if (loc\_ptr->info>item){

insert\_aux(loc\_ptr->lchild,item);}

else if (loc\_ptr->info<item){

insert\_aux(loc\_ptr->rchild,item);}

else{

cout<<"the item is already in the tree\n";}

}

void BST::insert\_Iterate(string item){

treenode \*parent, \*loc\_ptr = root;

treenode \*loc\_ptr2 = new treenode;

loc\_ptr2->lchild = loc\_ptr2->rchild = 0;

loc\_ptr2->info = item;

if (root == 0){

root = loc\_ptr2;

return;}

while(loc\_ptr != 0){

parent = loc\_ptr;

if(loc\_ptr->info==item){

cout<<"Duplicate in tree; do nothing!\n";

delete loc\_ptr2;

return;}

else if (loc\_ptr->info>item){

loc\_ptr = loc\_ptr->lchild;}

else if (loc\_ptr->info<item){

loc\_ptr = loc\_ptr->rchild;}

}

if (parent->info<item){

parent->rchild = loc\_ptr2;}

else{

parent->lchild = loc\_ptr2;}

}

treenode \* BST::search\_tree(string item){

return search\_tree\_aux(root, item);}

treenode \* BST::search\_tree\_aux(treenode \* loc\_ptr, string item){

if (loc\_ptr!=0) {

if(loc\_ptr->info==item){

return loc\_ptr;}

else if (loc\_ptr->info>item){

return search\_tree\_aux(loc\_ptr->lchild,item);}

else{

return search\_tree\_aux(loc\_ptr->rchild,item);}

}

else{

return loc\_ptr;}

}

treenode \* BST::search\_tree\_Iterate(string item){

treenode \* loc\_ptr = root;

while(loc\_ptr != 0){

if(loc\_ptr->info==item){

return loc\_ptr;}

else if (loc\_ptr->info>item){

loc\_ptr = loc\_ptr->lchild;}

else{

loc\_ptr = loc\_ptr->rchild;}

}

return loc\_ptr;}

void BST::del(string item){

del\_aux(root,item);}

void BST::del\_aux(treenode \* & loc\_ptr, string item){

if (loc\_ptr==0){

cout<<item<<" item not in tree,\n";}

else if (loc\_ptr->info > item){

del\_aux(loc\_ptr->lchild, item);}

else if (loc\_ptr->info < item){

del\_aux(loc\_ptr->rchild, item);}

else{

treenode \* ptr;

if (loc\_ptr->lchild == 0){

ptr=loc\_ptr->rchild;

delete loc\_ptr;

loc\_ptr=ptr;

}

else if (loc\_ptr->rchild == 0){

ptr=loc\_ptr->lchild;

delete loc\_ptr;

loc\_ptr=ptr;

}

else{

ptr=inorder\_succ(loc\_ptr);

loc\_ptr->info = ptr->info;

del\_aux(loc\_ptr->rchild, ptr->info);

}

}

}

treenode \* BST::inorder\_succ(treenode \* loc\_ptr){

treenode \*ptr=loc\_ptr->rchild;

while(ptr->lchild!=0){

ptr=ptr->lchild;}

return ptr;}

void BST::print\_tree(){

print\_tree\_aux(root);}

void BST::print\_tree\_aux(treenode \* loc\_ptr){

if (loc\_ptr!=0){

print\_tree\_aux(loc\_ptr->lchild);

cout << loc\_ptr->info<<endl;

print\_tree\_aux(loc\_ptr->rchild);

}}

BST::~BST(){

while (root!=0){

del(root->info);}}

int BST::count\_node(){

return count\_node(root);}

int BST::count\_node(treenode \* loc\_ptr){

if (loc\_ptr == 0){

return 0;}

else{

return 1 + count\_node(loc\_ptr->lchild) + count\_node(loc\_ptr->rchild);}}

int BST::count\_node(string r1, string r2){

return count\_node(root, r1, r2);}

int BST::count\_node(treenode \* r, string r1, string r2){

if (r == 0){

return 0;}

else if (r->info >= r1 && r->info <= r2){

return 1 + count\_node(r->lchild,r1,r2) + count\_node(r->rchild,r1,r2);}

else if (r->info > r2){

return count\_node(r->lchild,r1,r2);}

else if (r->info < r1){

return count\_node(r->rchild,r1,r2);}

else{

return 0;}

}

int main()

{

BST B;

//treenode \*root1=0, \*root2=0;

string s;

char ch;

string key3;

string key4;

string response;

string r1, r2;

cout<<"Enter command:\nc=count\ni=insert item\ns=search tree\nd=delete item\np=print tree\nr = count range\ne=exit\n";

cin>>ch;

cout<<endl;

while (ch!='e')

{

switch (ch)

{

case 'i' :cout<<"enter string: ";

cin>>s;

B.insert\_Iterate(s);

break;

case 's' :cout<<"enter word to search for: ";

cin>>s;

if (!B.search\_tree\_Iterate(s))

cout<<s<<"not in tree\n";

else

cout<<s<<" was found in the tree\n";

break;

case 'd' :cout<<"enter word to delete: ";

cin>>s;

B.del(s);

break;

case 'p' :cout<<"...printing tree...\n";

B.print\_tree();

break;

case 'c' :cout<<"...count nodes...\n";

cout<<"there are "<<B.count\_node()<<" nodes\n";

break;

case 'r' :cout<<"...count nodes in range...\n";

cout<<"enter range (string) r1 then r2:";

cin>>r1>>r2;

cout<<"there are "<<B.count\_node(r1,r2)<<" nodes\n";

break;

default:break;

}

cout<<"enter command, i=insert item,s=search tree,d=delete item,p=print tree,r = range, e=exit: ";

cin>>ch;

cout<<endl;

}

cout<<endl<<"No more binary tree...\n";

return 0;

}