//implementation file for the BST node and client BST

#include "Client Info BST.h"

Clients\_Info\_BST::Clients\_Info\_BST(){

//stores the data in the hash table

root = 0;

}

Clients\_Info\_BST::Clients\_Info\_BST(const Clients\_Info\_BST & copy){

//Copy Constructor

//Same thing as default constructor, but the input is the BST given in the argument

}

//^^^^^^^^^^^^^^^^^^^^^^^^^

Clients\_Info\_BST::~Clients\_Info\_BST(){

//Destructor

while (root!=0){

del(root->lastname);}

}

void Clients\_Info\_BST::Insert(const string & item){

cout << "Inside Clients\_Info\_BST Insert.\n";

Insert(root,item);

//Insert adds a new Client's information (into a BST\_Node) into the BST

}

//^^^^^^^^^^^^^^^^^^^^^^^^^ Modify so that it parses the string so that each bit of info is placed in the correct variable.

void Clients\_Info\_BST::Insert(BST\_Node \* & loc\_ptr, string item){

//Auxicilary function used by Insert above to allow recursion

if (loc\_ptr==0){

loc\_ptr = new BST\_Node;

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

loc\_ptr->lastname=item;

}

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

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

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

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

else{

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

}

void Clients\_Info\_BST::Remove(const string & item){

cout << "Inside Clients\_Info\_BST Remove.\n";

Remove(root, item);

//Remove deletes a BST\_Node that contains the specified client info from the BST if it is there

//otherwise a message should be printed stating so.

}

void Clients\_Info\_BST::Remove(BST\_Node \* & loc\_ptr, string item){

//Auxicilary function used by Removbe above to allow recursion

if (loc\_ptr==0){

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

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

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

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

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

else{

BST\_Node \* 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->lastname = ptr->lastname;

Remove(loc\_ptr->rchild, ptr->lastname);

}

}

}

void Clients\_Info\_BST::Update(const string & s){

cout << "Inside Clients\_Info\_BST Update.\n";

//Update modifies a client information given the first and last name if it is in the BST otherwise prints a message stating so

switch:

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

default:

}

//^^^^^^^^^^^^^^^^^^^^^^^^^

void Clients\_Info\_BST::Print(){

cout << "Inside Clients\_Info\_BST Print.\n";

print\_tree\_aux(root);

//Print outputs a BST, INORDER, to the display

}

void Clients\_Info\_BST::Print(BST\_Node \*){

//Auxicilary function used by Print above to allow recursion

if (loc\_ptr!=0){

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

cout << loc\_ptr->lastname<<"\t"<<loc\_ptr->firstname<<"\t"<<loc\_ptr->address<<"\t"<<loc\_ptr->phone<<"\n";

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

}}

BST\_Node \* Clients\_Info\_BST::Search(const string & item){

cout << "Inside Clients\_Info\_BST Search.\n";

return Search(root, item);

}

BST\_Node \* Clients\_Info\_BST::Search(BST\_Node \* loc\_ptr,string item){

//Auxicilary function used by Search above to allow recursion

if (loc\_ptr!=0) {

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

return loc\_ptr;}

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

return Search(loc\_ptr->lchild,item);}

else{

return Search(loc\_ptr->rchild,item);}

}

else{

return loc\_ptr;}

}

BST\_Node \* Clients\_Info\_BST::inorder\_succ(BST\_Node \*){

//Return pointer to inorder successor otherwise 0

BST\_Node \*ptr=loc\_ptr->rchild;

while(ptr->lchild!=0){

ptr=ptr->lchild;}

return ptr;

}

/\*void Clients\_Info\_BST::del\_aux(BST\_Node \* & 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);

}

}

}

void Clients\_Info\_BST::insert\_aux(BST\_Node \* & loc\_ptr, string item){

if (loc\_ptr==0){

loc\_ptr = new BST\_Node;

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

loc\_ptr->lastname=item;

}

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

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

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

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

else{

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

}

void Clients\_Info\_BST::print\_tree\_aux(BST\_Node \* loc\_ptr){

if (loc\_ptr!=0){

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

cout << loc\_ptr->lastname<<"\t"<<loc\_ptr->firstname<<"\t"<<loc\_ptr->address<<"\t"<<loc\_ptr->phone<<"\n";

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

}}\*/