#include <string>

#include <iostream>

#include <vector>

#include <fstream>

#include <sstream>

#include<cstdlib>

#include<iomanip>

using namespace std;

struct person //creates object for person

{

int SSN;

int bday;

string fn;

string ln;

int zip;

person(int ssn, int bd, string f\_name, string l\_name, int zc) //constructor

{

SSN = ssn;

bday = bd;

fn = f\_name;

ln = l\_name;

zip = zc;

}

void print()

{

cout << SSN <<" "<< bday<<" "<< fn<<" "<< ln<< " "<< zip<<endl;

}

void print\_old()

{

cout << "The oldest person is " << fn << " " << ln << " " << zip << endl;

}

};

struct node

{

person \* data;

node \* left;

node \* right;

node(person \* d)

{

data = d;

left = NULL;

right = NULL;

}

node \* find\_and\_delete(node \* N)

{

node \* current= N;

while(current && current->left != NULL)

{ current= current->left; }

return current;

}

person \* old\_age()

{

if(this == NULL)

return NULL;

person \* l\_age = left->old\_age();

person \* r\_age = right->old\_age();

person \* oldest = data;

if(l\_age != NULL)

{

if(l\_age->bday < oldest->bday)

oldest = l\_age;

}

if(r\_age != NULL)

{

if(r\_age->bday < oldest->bday)

oldest = r\_age;

}

//oldest->print\_old();

return oldest;

}

};

struct Tree

{

node \* root;

Tree()

{

root = NULL;

}

void add(person \* d, node \*& N);

void print\_tree(node \* N);

void find(node \* N, string f, string l);

void find\_zip(node \* N, int zip);

void find\_oldest();

//void delete\_node(node \* N, string f, string l);

node\* find\_for\_delete(node \*N);

node\* remove(node \*the\_one, string f, string l);

};

//creates tree

void Tree :: add(person \* d, node \*& N)

{

if(N == NULL) //tree is empty, we create new root

N = new node(d);

else if(d->ln > N->data->ln)

add(d, N->right);

else if(d->ln < N->data->ln)

add(d, N->left);

else if(d->ln == N->data->ln) //compares fn if ln are the same

{

if(d->fn > N->data->fn)

add(d, N->right);

else if(d->fn < N->data->fn)

add(d, N->left);

}

}

void Tree :: print\_tree(node \* N)

{

if(N == NULL)

return;

print\_tree(N->left);

N->data->print();

print\_tree(N->right);

}

void Tree :: find(node \* N, string f, string l)

{

while(N != NULL)

{

if(N->data->ln == l)

{

if(N->data->fn < f)

return find(N->right, f, l);

else if(N->data->fn > f)

return find(N->left, f, l);

else if(N->data->fn == f)

return N->data->print();

}

else if(N->data->ln < l)

return find(N->right, f, l);

else if(N->data->ln > l)

return find(N->left, f, l);

}

cout << "Sorry, person could not be found." <<endl;

}

void Tree :: find\_zip(node \* N, int zip)

{

if(N == NULL)

return;

find\_zip(N->left, zip);

if(N->data->zip == zip)

N->data->print();

find\_zip(N->right, zip);

}

void Tree:: find\_oldest()

{

person \* oldest\_person = root->old\_age();

oldest\_person->print\_old();

}

node \* Tree:: find\_for\_delete(node \* N) //looking for the one on the left most side

{

node\* current= N;

while(current && current->left != NULL)

{ current=current->left;

}

return current;

}

node \* Tree:: remove(node \*the\_one, string f, string l)

{

if(the\_one==NULL)

return the\_one;

if(l< the\_one->data->ln)

the\_one->left = remove(the\_one->left,f ,l);

else if(l > the\_one->data->ln)

the\_one->right = remove(the\_one->right, f, l);

else

{

if(f == the\_one->data->fn)

{ if(the\_one->left == NULL && the\_one->right == NULL)

return NULL;

else if(the\_one->left == NULL) //one child on right

{

node\* temp = the\_one->right;

delete the\_one;

return temp;

}

else if(the\_one->right == NULL)// one child on left

{

node \* temp= the\_one->left;

delete the\_one;

return temp;

}

node \* temp= find\_for\_delete(the\_one->right);

// replace the deleted one

the\_one->data->SSN = temp->data->SSN;

the\_one->data->bday = temp->data->bday;

the\_one->data->fn = temp->data->fn;

the\_one->data->ln = temp->data->ln;

the\_one->data->zip = temp->data->zip;

the\_one->right = remove(the\_one->right, temp->data->fn, temp->data->ln);

}

else

{

if(f < the\_one->data->fn)

the\_one->left = remove(the\_one->left, f, l);

else if (f > the\_one->data->ln)

the\_one->right = remove(the\_one->right, f, l);

}

}

return the\_one;

}

int read\_file()

{

Tree T;

ifstream fin;

fin.open("myfile.txt");

int ssn, bd;

string fn, ln;

int zip;

if (fin.fail())

{

cout << "Couldn't open the file\n";

exit(1);

}

else

{

int i=0;

while(i<10000)

{

if(fin.fail())

{

cout<<"Could not open file."<<endl;

break;

}

fin >> ssn >> bd >> fn >> ln >> zip;

person \* new\_d = new person(ssn, bd, fn, ln, zip);

T.add(new\_d, T.root);

i++;

}

}

fin.close();

while(true)

{

cout<<"Type command: "<<endl<<"Find - display all information on said person"<<endl<<"All - display the information of everyone in the Binary Tree"<<endl<<"Zip - displays the names of all people living in the given zip code"<<endl<<"Oldest - displays the name and zipcode of the oldest person" <<endl<<"Delete - Delete someone" <<endl<< "Exit - Exit out of this program." <<endl;

cout<<endl<<"-----------------------------------------------"<< endl;

string command;

cin>>command;

if(command=="Find")

{

cout<<"Who would you like to find? Enter first name."<<endl;

string first, last;

cin>>first;

cout<<"Enter last name."<<endl;

cin>>last;

T.find(T.root, first, last);

}

else if(command=="Exit")

break;

else if(command=="All")

{

T.print\_tree(T.root);

}

else if(command=="Zip")

{

cout<<"Enter desired zip code."<<endl;

int zc;

cin>>zc;

T.find\_zip(T.root, zc);

}

else if(command=="Oldest")

T.find\_oldest();

else if(command=="Delete")

{ cout<<"Who would you like to delete from this database?"<<endl;

string first\_name, last\_name;

cout<<"Enter first name." <<endl;

cin>>first\_name;

cout<<"Enter last name."<<endl;

cin>>last\_name;

T.remove(T.root, first\_name, last\_name);

cout<<"The person "<<first\_name<<" "<<last\_name<<" has been deleted from your Binary Tree."<<endl;

}

else

cout << "Command not valid." << endl;

cout<<"-----------------------------------------------"<<endl;

}

return 0;

}

int main()

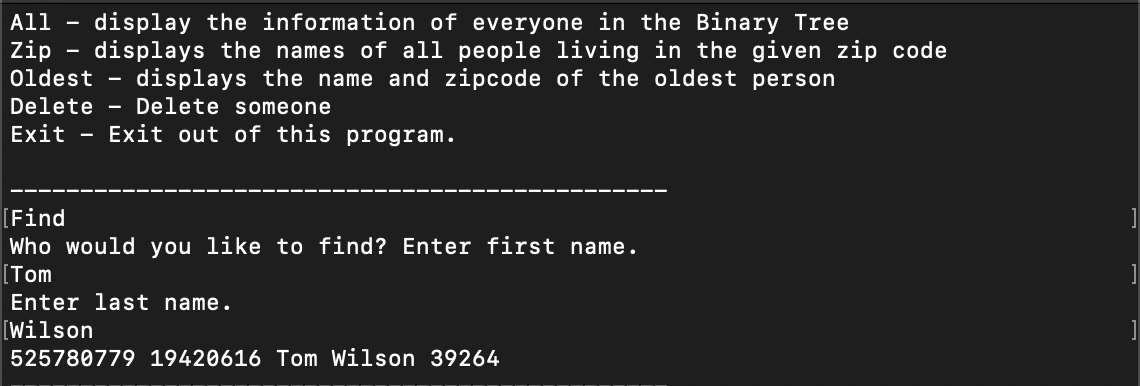
{

read\_file();

return 0;

}

Find Function



All

Text

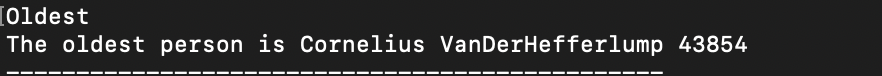
Description automatically generated with medium confidence

Zip

Text

Description automatically generated with medium confidence

Oldest



Delete

Text

Description automatically generated

Tom Wilson is no longer findable