|  |  |
| --- | --- |
| **Total Marks:** | **1** |
| **Obtained Marks:** |  |

Data Structures and Algorithms

(CSCL 2202)

# Lab Task #10

# Last date of submission: 4th January, 2022.

# Submitted to: *Tehreem Saboor.*

# Student name: Hammad Ahmed Qureshi

# Reg. Number: 2112114

# Q no 1: Implement a city database using a BST to store the database records. Each database record contains the name of the city (a string of arbitrary length) and the coordinates of the city expressed as integer x- and y-coordinates. The BST should be organized by city name. Your database should allow records to be inserted, deleted by name or coordinate, and searched by name or coordinate. Another operation that should be supported is to print all records within a given distance of a specified point. Collect running-time statistics for each operation. Which operations can be implemented reasonably efficiently (i.e., in Θ(log n) time in the average case) using a BST? Can the database system be made more efficient by using one or more additional BSTs to organize the records by location?

# Code:

#include <iostream>

#include <string>

#include <cmath>

using namespace std;

struct city{

string name;

int x;

int y;

city \*left;

city \*right;

bool deleted;

};

class database{

private:

city \*root;

void insert(city \*&temp, string name, int x, int y){

if(temp == NULL){

temp = new city();

temp->name = name;

temp->x = x;

temp->y = y;

temp->left = NULL;

temp->right = NULL;

temp->deleted = false;

}

else{

if(name < temp->name){

insert(temp->left, name, x, y);

}

else if(name > temp->name){

insert(temp->right, name, x, y);

}

else{

if(temp->deleted == true){

temp->deleted = false;

temp->x = x;

temp->y = y;

}

else{

insert(temp->right, name, x, y);

}

}

}

}

city \*findByName(city \*temp, string name){

if(temp != NULL){

if(temp->name == name) return (temp->deleted == false ? temp : findByName(temp->right, name));

else if(name < temp->name) return findByName(temp->left, name);

else if(name > temp->name) return findByName(temp->right, name);

}

else{

return NULL;

}

}

city \*findByCoordinates(city \*temp, int x, int y){

if(temp == NULL){

return NULL;

}

else{

city \*ret = NULL;

if(temp->x == x && temp->y == y){

if(temp->deleted == false){

ret = temp;

}

}

if(ret == NULL){

ret = findByCoordinates(temp->left, x, y);

}

if(ret == NULL){

ret = findByCoordinates(temp->right, x, y);

}

return ret;

}

}

void print(city \*temp, int x, int y, double d){

print(temp->left, x, y, d);

if(temp->deleted == false){

double distance = sqrt(pow(temp->x - x, 2) + pow(temp->y - y, 2));

if(distance < d){

cout << temp->name << "at (" << temp->x << ", " << temp->y << ")" << endl;

}

}

print(temp->right, x, y, d);

}

public:

database(){

root = NULL;

}

void insert(string name, int x, int y){

insert(root, name, x, y);

}

void searchByName(string name){

city \*found = findByName(root, name);

if(found == NULL){

cout << "City not found" << endl;

}

else{

cout << "City " << found->name << " found at (" << found->x << ", " << found->y << ")" << endl;

}

}

void searchByCoordinates(int x, int y){

city \*found = findByCoordinates(root, x, y);

if(found == NULL){

cout << "City not found" << endl;

}

else{

cout << "City " << found->name << " found at (" << found->x << ", " << found->y << ")" << endl;

}

}

void deleteByName(string name){

city \*found = findByName(root, name);

if(found == NULL){

cout << "City not found" << endl;

}

else{

cout << "City " << found->name << " found at (" << found->x << ", " << found->y << ") has been deleted" << endl;

found->deleted = true;

}

}

void deleteByCoordinates(int x, int y){

city \*found = findByCoordinates(root, x, y);

if(found == NULL){

cout << "City not found" << endl;

}

else{

cout << "City " << found->name << " found at (" << found->x << ", " << found->y << ") has been deleted" << endl;

found->deleted = true;

}

}

void printInDistance(int x, int y, double d){

print(root, x, y, d);

}

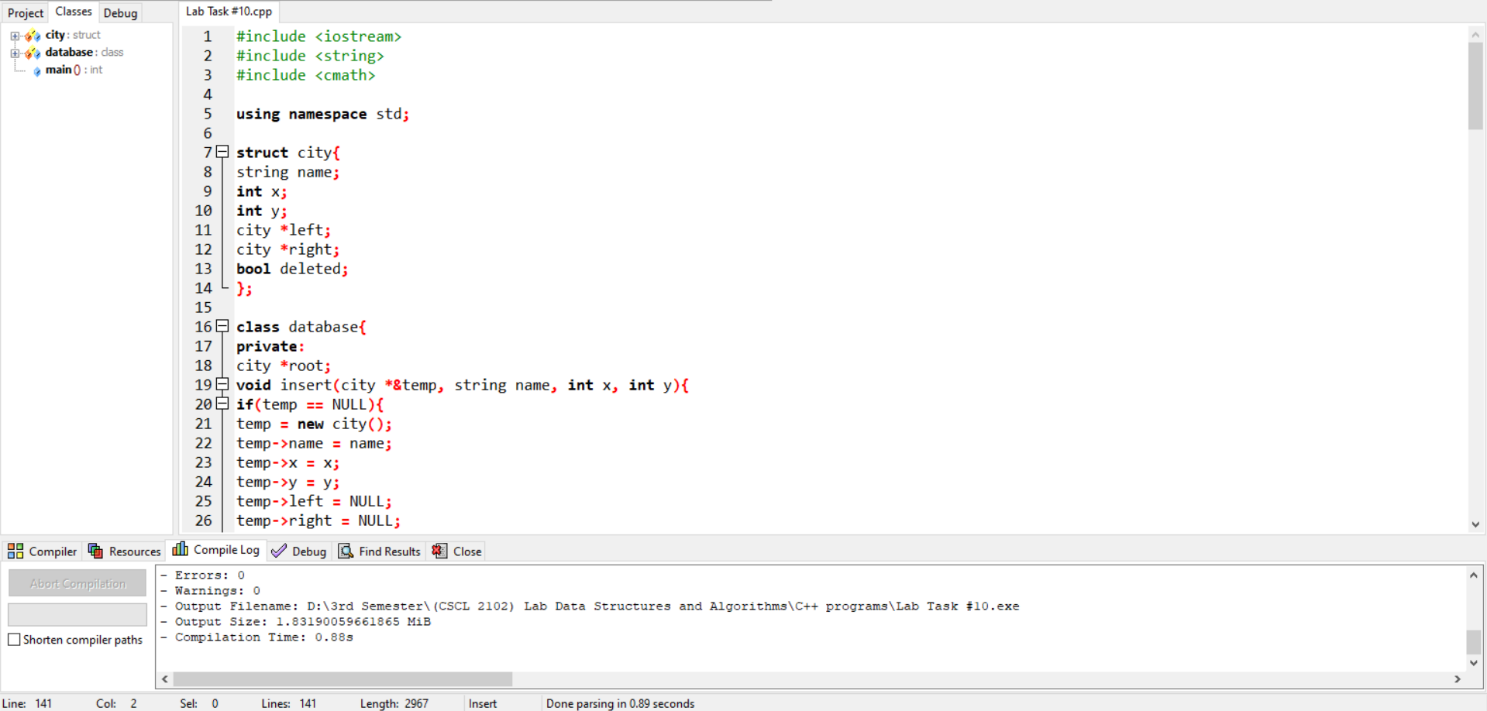
};

int main(){

return 0;

}

# Console Screenshots:



# Output Screenshots:

