```
Script started on 2021-02-24 12:08:10-0600
m_sadaf1@ares:~$ pwd
/home/students/m_sadaf1
m_sadaf1@ares:~$ cat map.info
Name: Madiha Sadaf
Class: CSC122 W01

Project: "Map it out"
Option: Added a third class to manage the city list.

Level: 4
Level: +2
Total Level: 6

Description:

This program allows a user to to enter a city's name and
coordinates to calculate the distance between them on a map.

m_sadaf1@ares:~$ cat city_main.cpp
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include "city.h"
#include "city_list.h"
#include "input_prot.h"
#include "point.h"

using namespace std;

// Strings

const string prompt = "Select an option: ";
const string error_prompt = "Invalid input.";
const string menu =
        " Choose your option: \n\n"
        "1: Enter city information \n"
        "2: Calculate distance between two cities \n"
        "3: Print all cities \n"
        "4: Quit \n";

const string menu_choices =
        "1Ee"     // Enter city
        "2Cc"     // Calculate distance
        "3Pp"     // Print cities
        "4Qq";    // Quit program

int main()
{
    cout << "\t\t\tWelcome to the Distance Calc Program! \n";
    char choice;
    bool done = false;
    vector<char> input_list;
    vector<char> yes_no;
    input_char("YyNn", yes_no);
    input_char(menu_choices, input_list);
    City_list city_list;

    while (!done)
    {
        cout << menu << endl;
        input_prot(choice, input_list, prompt, error_prompt);
        switch (choice)
        {
            case '1': case 'E': case 'e':
            {
                Point location;
                string name;

                cout << "Enter city name: ";
                cin.clear();
                cin.ignore();
                getline(cin, name);
                cout << "Enter coordinates ([x-coordinate],"
                        "[y-coordinate]): ";
                location.input();

                City c(location, name);
                bool added = city_list.add_city(c);
                if (added)
                {
                    cout << "'" << name << " added.";
                }
                else
                {
                    cout << "List is complete.\n";
                    input_prot(choice, yes_no,
                            "Would you like to overwrite"
                            " a city? Y/N: ", error_prompt);
                    if (choice == 'n' || choice == 'N')
                    {
                        cout << " Cancelled.\n";
                    }
                    else
                    {
                        city_list.list_cities("");
                        long pos;
                        input_prot(pos, 1, city_list.
                                get_list_length(), "Choose "
                                "a city to overwite: ",
                                error_prompt);
                        pos--;
                        cout << city_list.get_city(pos)
                                .get_name() <<
                                " overwritten.";
                        city_list.set_city(pos, c);
                    }
```

```
    }
}
    break;

case '2': case 'C': case 'c':
{
    if ((city_list.get_list_length()) < 2)
    {
        cout << "You need at least two cities to "
                "calculate the distance.\n";
    }
    else if ((city_list.get_list_length()) == 2)
    {
        cout << "Distance between the two cities "
                "is: ";
        cout << city_list.get_city(0)
                .get_distance(city_list.get_city
                (1));
    }
    else
    {
        city_list.list_cities("");
        long pos_1;
        input_prot(pos_1, 1, city_list
                .get_list_length(), "Enter the "
                "POSITION (not coordinates) of "
                "the first city: ",
                error_prompt);
        // bool done = false;
        long pos_2;
        while (!done)
        {
            input_prot(pos_2, 1, city_list
                .get_list_length(), "Enter the "
                "POSITION (not coordinates) of "
                "the second city: ",
                error_prompt);
            if (pos_2 == pos_1)
            {
                cout << "You cannot input the "
                        "same position.\n";
            }
            else
            {
                done = true;
            }
        }
        cout << "Distance: ";
        cout << (city_list.get_city(pos_1 - 1))
                .get_distance(city_list.get_city
                (pos_2 - 1));
    }
}
    break;
```

```
            case '3': case'P': case'p':
            {
                city_list.list_cities("No cities inputted.");
            }
                break;

            case '4': case 'Q': case'q':
                {
                    cout << "Bye!\n";
                    done = true;
                }
                    break;

                default:
                {
                    cout << error_prompt;
                }
            } // switch end
        } // while (!done) end

        return 0;

    }

m_sadaf1@ares:~$ cat city.cpp
#include "point.h"
#include "city.h"
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

// Accessors

Point City::get_location() const
{
    return location;
}

string City::get_name() const
{
    return name;
}

double City::get_distance(const City & other) const
{
    return location.distance(other.location);
}

// Mutators

void City::set_location(Point c_loc)
```

```
{
    location = c_loc;
}

void City::set_name(string c_name)
{
    name = c_name;
}

m_sadaf1@ares:~$ cat input_prot.cpp
#include <iostream>
#include <string>
#include "input_prot.h"
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

void input_prot(long & input, const string & prompt,
        const string & error_prompt)
{
    bool complete = false;
    while (complete == false)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n') //Any extra character
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (cin.fail())
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else
        {
            complete = true;
        }
    }
}

void input_prot(long & input, const long & bounce,
        const string & prompt, const string & error_prompt,
        const bool & bounce_max)
{
    bool complete = false;
    while (complete == false)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n')
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (cin.fail())
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
                    complete = false;
        }
        else if (!bounce_max) // Value is min
        {
            if (input < bounce)
            {
                cerr << error_prompt << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
                complete = false;
            }
            else
            {
                complete = true;
            }
        }
        else // Value is max
        {
            if (input > bounce)
            {
                cerr << error_prompt << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
                complete = false;
            }
            else
            {
                complete = true;
            }
        }
    }
}
```

```
void input_prot(long & input, const long & min,
        const long & max, const string & prompt,
        const string & error_prompt)
{
    bool complete = false;
    while (complete == false)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n')
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (cin.fail())
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
                    complete = false;
        }
        else if (input < min || input > max)
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else
        {
            complete = true;
        }
    }
}

void input_prot(long & input, const vector<long> & list,
        const string & prompt, const string & error_prompt)
{
    bool complete = false;
    while (!complete)
    {
        input_prot(input, prompt, error_prompt);
        for (vector<long>::size_type pos = 0; pos <
                list.size(); pos++)
        {
            if (input == list[pos]) // If found in list, search is over
            {
                complete = true;
            }
        }
        if (!complete)
        {
            cerr << error_prompt << endl;
        }
    }
}

void input_prot(double & input, const string & prompt,
        const string & error_prompt)
{
    bool complete = false;
    while (complete == false)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n')
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (cin.fail())
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else
        {
            complete = true;
        }
    }
}

void input_prot(double & input, const double & bounce,
        const string & prompt, const string & error_prompt,
        const bool & bounce_max)
{
    bool complete = false;
    while (complete == false)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n')
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
```

```
            complete = false;
        }
        else if (cin.fail())
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (!bounce_max)
        {
            if(input < bounce)
            {
                cerr << error_prompt << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
                complete = false;
            }
            else
            {
                complete = true;
            }
        }
        else  //Value is max
        {
            if (input > bounce)
            {
                cerr << error_prompt << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
                complete = false;
            }
            else
            {
                complete = true;
            }
        }
    }
}

void input_prot(double & input, const double & min,
        const double max, const string & prompt,
        const string & error_prompt)
{
    bool complete = false;
    while (complete == false)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n')
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (cin.fail())
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else if (input < min || input > max)
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else
        {
            complete = true;
        }
    }
}

void input_prot(double & input, const vector<double> & list,
        const string & prompt, const string & error_prompt)
{
    bool complete = false;
    while (complete == false)
    {
        input_prot(input, prompt, error_prompt);
        for (vector<double>::size_type pos = 0; pos <
                list.size(); pos++)
        {
            if (fabs((input - list[pos])) < 1e-6) // If found in list, search is ov
            {
                complete = true;
            }
        }
        if (!complete)
        {
            cerr << error_prompt << endl;
        }
    }
}

void input_prot(char & input, const string & prompt,
        const string & error_prompt)
{
```

```
    bool complete = false;
    while(!complete)
    {
        cout << prompt;
        cin >> input;
        if (cin.peek() != '\n')
        {
            cerr << error_prompt << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::
                    max(), '\n');
            complete = false;
        }
        else
        {
            complete = true;
        }
    }
}

void input_prot(char & input, const vector<char> & list,
        const string & prompt, const string & error_prompt)
{
    bool complete = false;
    while(complete == false)
    {
        input_prot(input, prompt, error_prompt);
        for (vector<char>::size_type pos = 0;
                pos < list.size(); pos++)
        {
            if (input == list[pos])
            {
                complete = true;
            }
        }
        if (!complete)
        {
            cerr << error_prompt << endl;
        }
    }
}

// Adds string to vector

void input_char(const string & input,
        vector<char> & list)
{
    for (string::size_type pos = 0; pos < input.length();
            pos++)
    {
        list.push_back(input[pos]);
    }
}
```

```
m_sadaf1@ares:~$ cat city_list.cpp
#include <iostream>
#include "city.h"
#include "city_list.h"

using namespace std;

City City_list::get_city(vector<City>::size_type
        index) const
{
    return city_list[index];
}

void City_list::list_cities(const string & error) const
{
    if (city_list.size() > 0) // Checks if list is not empty
    {
        for (vector<City>::size_type pos = 0; pos <
                city_list.size(); pos++)
        {
            cout << pos + 1 << ": ";
            cout << get_city(pos).get_name() << " at ";
            (city_list[pos].get_location()).output();
            cout << ".\n";
        }
    }
    else    // List is empty
    {
        cerr << error << endl;
    }
}

void City_list::set_city(vector<City>::size_type index,
        const City & other_city)
{
    city_list[index] = other_city;
}

bool City_list::add_city(const City & other_city)
{
    bool added;
    if (city_list.size() < max_cities)
    {
        city_list.push_back(other_city);
        added = true;
    }
    else
    {
        added = false;
    }
    return added;
}

vector<City>::size_type City_list::get_list_length() const
```

```
{
    return city_list.size();
}

m_sadaf1@ares:~$ cat city_point.cpp
#include <iostream>
#include <cmath>
#include "point.h"

using namespace std;

// Reads 2D point notation (x,y)

void Point::input(void)
{
    char read;
    cin >> read >> x >> read >> y >> read;
    return;
}

// Outputs 2D point notation (x,y)

void Point::output(void) const
{
    cout << '(' << x << ", " << y << ')';
    return;
}

// Calc distance between 2 2D points

double Point::distance(const Point & other) const
{
    return sqrt(pow(x - other.x, 2.0) +
            pow(other.y -y, 2.0));
}

// Calc midpoint between 2 2D points

Point Point::midpoint(const Point & other) const
{
    return Point((x + other.x) / 2.0, (other.y + y) / 2.0);
}

// Set coordinates to values

void Point::set_x(double new_x)
{
    x = new_x; // no error checking, anything is valid
    return;
}

void Point::set_y(double new_y)
{
    y = new_y;
    return;
}

// Creates a point flipped (x-axis)

Point Point::flip_x(void) const
{
    return Point(x, -y);
}

// Creates a point flipped (y-axis)

Point Point::flip_y(void) const
{
    return Point(-x, y);
}

// Creates a point shiftes along the x-axis

Point Point::shift_x(double move) const
{
    return Point(x + move, y);
}

// Creates a point shiftes along the y-axis

Point Point::shift_y(double move) const
{
    return Point(x, y + move);
}

m_sadaf1@ares:~$ cat city.h
#include <string>
#include "point.h"
#ifndef CITY_C_ADDED
#define CITY_C_ADDED

// 2D class

class City
{
    Point location;
    std::string name;

public:

    // Constructors

    City() : location(), name() {}
    City(Point c_loc) : location(c_loc), name() {}
    City(std::string c_name) : location(), name(c_name) {}
    City(Point c_loc, std::string c_name) :
            location(c_loc), name(c_name) {}
```

```
    // Accessors

    Point get_location() const;
    std::string get_name() const;
    double get_distance(const City & other) const;
    void display_city();

    // Mutators

    void set_location(Point c_loc);
    void set_name(std::string c_name);
};


#endif // CITY_C_ADDED

m_sadaf1@ares:~$ cat input_prot.h
#ifndef INPUT_H
#define INPUT_H

#include <string>
#include <cstdlib>
#include <vector>

// Checks to see if input is long

void input_prot(long & input, const std::string & prompt,
const std::string & error_prompt);

// Max or Min

void input_prot(long & input, const long & bounce,
        const std::string & prompt, const std::string & error_prompt,
        const bool & bounce_max);

//Max and Min

void input_prot(long & input, const long & min,
        const long & max, const std::string & prompt,
        const std::string & error_prompt);

// Checks to see if input is in list of longs

void input_prot(long & input, const std::vector<long>
        & list, const std::string & prompt,
        const std:: string & error_prompt);

// Checks to see if input is double

void input_prot(double & input, const std::string & prompt,
        const std::string & error_prompt);

// Max or Min

void input_prot(double & input, const double & bounce,
        const std::string & prompt, const std::string &
        error_prompt, const bool & bounce_max);

// Max and Min

void input_prot(double & input, const double & min, const
        double max, const std:: string & prompt,
        const std::string & error_prompt);

// Checks to see if input is in list of doubles

void input_prot(double & input, const
        std::vector<double> & list_input,
        const std::string & prompt,
        const std::string & error_prompt);

// Checks to see if input is char

void input_prot(char & input, const std::string & prompt,
        const std:: string & error_prompt);

// Checks to see if input is in list of chars

void input_prot(char & input, const std:: vector<char>
        & list, const std::string & prompt,
        const std:: string & error_prompt);

// Adds string to vector

void input_char(const std:: string & input,
        std::vector<char> & list);

#endif /*INPUT_H*/

m_sadaf1@ares:~$ cat city_list.h
#include<vector>
#include "city.h"
#ifndef city_list_c
#define city_list_c

const short max_cities = 3;

class City_list
{
    std::vector<City> city_list;

public:
    // Constructors
    City_list() : city_list() {}

    // Accessors
    void list_cities(const std::string & error) const;
    City get_city(std::vector<City>::size_type index) const;
```

```
    std::vector<City>::size_type get_list_length() const;

    // Mutators
    void set_city(std::vector<City>::size_type index, const
            City & other_city);
    bool add_city(const City & other_city);

};

#endif /*city_list_c*/


m_sadaf1@ares:~$ cat point.h
#ifndef POINT_CLASS_INCLUDED
#define POINT_CLASS_INCLUDED

// A 2D point class

class Point
{
    double x, // x coordinate
           y; // y coordinate

public:
    Point(void) :x(0), y(0) {}
    Point(double new_x, double new_y):x(new_x), y(new_y) {}
    Point(const Point & p): x(p.x), y(p.y) {}

    void output(void) const;
    void input(void);

    double distance(const Point & other) const;
    Point midpoint(const Point & other) const;

    double get_x(void) const { return x; }  // accessors
    double get_y(void) const { return y;}

    void set_x(double new_x);   // mutators
    void set_y(double new_y);

    Point flip_x(void) const;  // New flipped point
    Point flip_y(void) const;  // Specified axis

    Point shift_x(double move) const;  // New point
    Point shift_y(double move) const;  // Shifted move in
                                       // the given direction
};

#endif

m_sadaf1@ares:~$ cat citypwdcat citpwdCPP again input_prot
cat pwdCPP again input_prot
cat pwdCPP city city_input_prot city_list city_main city_point
city.cpp...
```

```
city_input_prot.cpp...
city_list.cpp...
city_main.cpp***
city_point.cpp...


m_sadaf1@ares:~$ ./city_main.out
                        Welcome to the Distance Calc Program!
 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 2
You need at least two cities to calculate the distance.
 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 3
No cities inputted.
 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Palatine
Enter coordinates ([x-coordinate],[y-coordinate]): (42.1103, 88.0342)
'Palatine added. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Des Plaines
Enter coordinates ([x-coordinate],[y-coordinate]): (42.0334, 87.8834)
'Des Plaines added. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
```

```
Enter city name: Schaumburg
Enter coordinates ([x-coordinate],[y-coordinate]): (42.0334, 88.0834)
'Schaumburg added. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Palatine
Enter coordinates ([x-coordinate],[y-coordinate]): (42.1103, 88.0342)
List is complete.
Would you like to overwrite a city? Y/N: y
1: Palatine at (42.1103, 88.0342).
2: Des Plaines at (42.0334, 87.8834).
3: Schaumburg at (42.0334, 88.0834).
Choose a city to overwite: 3
Schaumburg overwritten. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Schaumburg
Enter coordinates ([x-coordinate],[y-coordinate]): (42.0334, 88.0834)
List is complete.
Would you like to overwrite a city? Y/N: n
 Cancelled.
 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 2
1: Palatine at (42.1103, 88.0342).
2: Des Plaines at (42.0334, 87.8834).
3: Palatine at (42.1103, 88.0342).
Enter the POSITION (not coordinates) of the first city: 1
Enter the POSITION (not coordinates) of the second city: 1
You cannot input the same position.
Enter the POSITION (not coordinates) of the second city: 1
You cannot input the same position.
Enter the POSITION (not coordinates) of the second city: 3
Distance: 0m_sadaf1@ares:~$ ./city_main.out
                        Welcome to the Distance Calc Program!
 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Palatine
Enter coordinates ([x-coordinate],[y-coordinate]): (42.1103, 88.0342)
'Palatine added. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Des Plaines
Enter coordinates ([x-coordinate],[y-coordinate]): (42.0334, 87.8834)
'Des Plaines added. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 2
Distance between the two cities is: 0.169276 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 3
1: Palatine at (42.1103, 88.0342).
2: Des Plaines at (42.0334, 87.8834).
 Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: 1
Enter city name: Palatine
Enter coordinates ([x-coordinate],[y-coordinate]): 29732
(4, 6)
'Palatine added. Choose your option:

1: Enter city information
2: Calculate distance between two cities
3: Print all cities
4: Quit

Select an option: Invalid input.
Select an option: 2
```

```
1: Palatine at (42.1103, 88.0342).
2: Des Plaines at (42.0334, 87.8834).
3: Palatine at (9732, 4).
Enter the POSITION (not coordinates) of the first city: 9
Invalid input.
Enter the POSITION (not coordinates) of the first city: 3
Enter the POSITION (not coordinates) of the second city: 2
Distance: 9690.33m_sadaf1@ares:~$ exit
exit

Script done on 2021-02-24 12:19:28-0600
```