MTH 3300
Spring 2017
E. Fink
May 2, 2017

Homework 9

Directions: Write the following 2 programs, and submit your source code to me via Blackboard, using the template files I have provided. You should send only the source code, in cpp format; do NOT send your .exe files. READ THE INSTRUCTIONS on how to submit your work in the Course Documents section of Blackboard.

This programming assignment is meant to give practice on vectors and object-oriented programming.

---

1) entrances.cpp

It's finally time to write the subway program from the first day of class! Hopefully this will be easy at this point.

The file `entrancenames.txt` contains roughly 1400 lines, each containing the name of a subway station entrance. The files `coords.txt` and `lines.txt` are the same length as `entrancenames.txt`, and contain the subway lines that service those stations and the Latitude and Longitude of the entrance on the corresponding lines. For example, the fourth lines of all the files correspond to each other – so that there is a station entrance called `Clinton St & Montague St At Nw Corner` that is located at Latitude 40.6944 and Longitude -73.9925 which is serviced by the `2-3-4-5-N-R` lines.

Your program should read the contents of these files into 4 **vectors**! Specifically, one should contain all the Latitudes (the first column in the `coords.txt` file), one should contain all the Longitudes (the second column in the `coords.txt` file), one should contain all the entrance names (from the `entrancenames.txt` file), and one should contain the corresponding subway lines (from the `lines.txt` file). Do **not** use plain arrays, and do not assume that the files are of a fixed length (because I might use different data files when I test).

Then, ask the user to enter a Latitude and Longitude. Compute the distance between this point and each of the subway stations, using the scheme from HW3 (the "Parks" problem); use a function for this!

Keep track of which station is the least distance. (Hint: keep track of which station contains the closest entrance – is it the first entrance, the second entrance, the 325th entrance? That number is what you will need to keep track of.) Then, print out the name of the closest entrance, the lines that service that station, and the distance to the input location.

Sample input: if you input a Latitude of 40.66 and a Longitude of -74.0, the closest station should be `4th Ave & 25th St At Sw Corner (To Bay Ridge And Coney Island Only)`, serviced by the `D-N-R` lines, which is a distance of approximately 0.162 kilometers away.

Specifications: your program must

- read the data in the files into four **vectors**. Do *not* assume that the vectors are of any fixed length (i.e., use `.push_back()` to fill in the vectors).
- ask the user for a Latitude and Longitude.
- find the closest station, using the distance computation scheme you used in HW3, utilizing a function.
- print out the name of the closest entrance, the lines that stop there, and the distance from the Latitude and Longitude that were input.

Bonus: for a challenge, try finding the TEN nearest station entrances – I encourage everyone to try this! For a more tedious challenge, you can try doing the same problem using the single `subwayentrance.csv` file instead of the three individual `.txt` files – you might have to look up how to use `getline` with a *delimiter*. (Also, if you use the .csv file, your the distance for your test value will be slightly different than 0.162 kms, since the .csv contains more decimal places.)

---

2) tictactoe.cpp

Question: imagine that you play Tic-Tac-Toe randomly. Specifically, X goes first, placing an X on a random square. Then O goes, placing an `O` on one of the eight remaining squares. Then X goes again, placing an `X` on one of the seven remaning squares randomly. Then O goes again, etc., until one player gets three in a row. What is the probability that X wins this random game of Tic-Tac-Toe?

Create a class called `TicTacBoard` whose objects represent Tic-Tac-Toe boards. The underlying data will be 3 x 3 arrays of `char`s, each of whose entries will be either `'x'`, `'o'` or `' '`. The first two should be LOWERCASE; the last one, which will represent an unused square, is two single quotes with a space between them (the space character).

Specifically, you should implement your class to the following specifications. I suggest you deal with the members and functions in the order I write them.

The class should have the following **private** data member:

1. `board`, which is a 3x3 array of `char`s. This will represent the state of the board at any given time.

The class should also have the following member functions, all **public** unless otherwise noted:

2. a constructor with no parameters. This should simply initialize all of the entries to be `' '` – that is, the entries should be the space character, which is two single quotes with a space between them. (So, the board starts out as blank.)
3. `board_print`, which should take no outside inputs and return nothing. This should print out the values held in `board` in a reasonably pretty way, with an extra `endl` after the board is printed.
4. `set`, which should take two `int`s and a `char` as input, and change the entry at the given position to the given character. For instance, if I have a `TicTacBoard` object named `myGame`, then

   ```
   myGame.set(0,2,'x');
   ```

   will set the entry in the upper right corner to be `x` (because the upper right corner is the first row, third column, which corresponds to `board[0][2]` with zero-based indexing).
5. `occupied`, which should take two `int`s and return a `bool`. This function should check if the given position has already been played. For example,

   ```
   myGame.occupied(0,2);
   ```

   should return `true` if `board[0][2]` is either `'x'` or `'o'`, and it should return `false` if `board[0][2]` is `' '`.
6. `row_full`, which should be **PRIVATE**, an which should take an `int` and a `char`, and return a `bool`. This function should check if the row corresponding to the given `int` is entirely occupied by the given `char`. For example,

   ```
   myGame.row_full(2,'x');
   ```

   should return `true` if each entry in the third row of `board` is `'x'` (that is, if `board[2][0]`, `board[2][1]` and `board[2][2]` are all `'x'`).
7. `col_full`, which should be **PRIVATE**, an which should take an `int` and a `char`, and return a `bool`. This should be the same as the last function, except for columns instead of rows.
8. `diag_full`, which should be **PRIVATE**, an which should take a `char`, and return a `bool`. Sort of similar to the last two, this should check whether either of the diagonals contain the given letter all the way down: that is, it should return `true` if either `board[0][0]`, `board[1][1]`, and `board[2][2]` all contain the input `char`, or if `board[0][2]`, `board[1][1]`, and `board[2][0]` all contain the input `char`.
9. `win_for`, which should take a `char` as input and return a `bool`. This should take the given `char`, and return `true` if that `char` is in winning position: if there is a row, column or diagonal for which each entry contains that `char`. The function should return `false` otherwise.
10. `random_move`, which should take a `char` and return nothing. This function should play as the input `char`: it should take the board, and try to place the given `char` in any position that isn't already occupied with an `x` or an `o`. So,

    ```
    myGame.random_move('x');
    ```

    tries to put an `'x'` into any of the unoccupied squares, each one with equal probability. This is slightly tricky. Suggestion: pick two random numbers representing row and column; check if that square is occupied; if it isn't place the move, and if it is, pick two new numbers and start over.

I've put some test code in my `main()` function – please leave it in your final submission. **I strongly suggest that you build this class one component at a time, in the order I suggested, and then run the relevant portions of my test code, commenting out the parts that correspond to what you haven't implemented yet.**

After you write the class well enough so that all the test code works, you should attempt to answer the question I posed before: if two players play Tic Tac Toe by making random moves, what is the probability that the first player, X, is the winner?

The general strategy is: create a loop that runs, say, 1000000 times. Each time through, declare a `TicTacBoard`, use the member functions to simulate a game, and keep track of how many of those 1000000 games is won by X.

Specifications: your program must

- contain a definition of the class `TicTacBoard` that includes the 10 functions listed above, and that makes my test code run properly.
- create code that uses the `TicTacBoard` class to simulate 1000000 randomly-played Tic Tac Toe games, and report the number of games that were won by X.

(And, of course, you can also actually make a playable Tic Tac Toe game with this class. Duh.)