SIENAcollege
Computer Science

Computer Science 301
C Programming in Unix
Siena College
Fall A 2024

Assessment 2: METAL Vertex Search

Due: 9:00 AM, Wednesday, September 25, 2024

We're finally ready for your first programming assignment. Please review the syllabus description of the expectations for a programming assignment as opposed to a practice program.

You may work alone or in groups of size 2 or 3 on this assessment. However, in order to make sure you learn the material and are well-prepared for the exams, those who work in a group should either collaborate closely while completing the program or work through the program individually then it them within your group to agree on a solution. In particular, the "you do this and I'll do that" approach is sure to leave you unprepared for upcoming tasks and the exams, and is prohibited.

Learning goals:

1. To gain experience working with a larger C program.
2. To practice with file I/O, pointers, and `structs` in C.

Getting Set Up

In Canvas, you will find a link to follow to set up your GitHub repository, which will be named `vsearch-assessmentt-yourgitname`, for this assessment. Only one member of the group should follow the link to set up the repository on GitHub, then others should request a link to be granted write access.

All GitHub repositories must be created with all group members having write access and all group member names specified in the `README.md` file by 11:59 PM, Friday, September 20, 2024. This applies to those who choose to work alone as well!

Development Process Expectations and Reporting

Important: in order to encourage a step-by-step problem-solving approach and good software development practices, you must have no fewer than 5 commits of partially-working versions of this program, with good commit messages (e.g., "reading input file correctly", "first working version of the waypoint structure", etc.). A penalty up to 50% for this part of the assignment will be applied if these commits are not indicated below and/or do not demonstrate an appropriate development process.

Please add the 5 commits you wish to have considered for this requirement to the table that you will find in your repository's `README.md` file. To find the information to include in this table, go to you repository on GitHub. Then go to its commits page. At the end of each commit listed there, you will see an icon that looks like this: <>. This is a link to the GitHub repository as it

existed after that commit. Copy that link and the corresponding commit message into the table in the `README.md` file.

Implementing a Vertex Extremes Search

As many of you have done in the past, we'll be working with some data from the METAL project. In fact, you might have already written code to solve the "problem" we're looking at in Java for an earlier class.

METAL provides a collection of graph data files that represent various highway systems. Each of the "`.tmg`" files is in the format described on METAL's "Graph File Formats" page. You'll want to download a few smaller and a few larger data files, and should choose "collapsed" or "simple" format graphs. The "traveled" format graphs have additional information not needed for this task. You can use METAL's Highway Data Examiner to run the Vertex Extremes algorithm visualization to see how the algorithm works, and to check your answers, if you'd like.

Write a program with its `main` function in `extremes.c` that reads the "waypoints" portion of a graph file whose file name is specified as a command-line parameter, and then finds the easternmost, westernmost, northernmost, and southernmost waypoints in the file. For each "winner", print out all of that waypoint's information: its label, its latitude and its longitude. Handle ties by keeping only the first waypoint at a given extreme latitude or longitude that you encounter. Include a `Makefile` that builds your program.

Notes:

- You can safely ignore the "road segments" portion of the input file for this assignment.
- You can safely assume that no waypoint label is longer than 256 characters.
- By including the `\n` in your `fscanf` format string, you can automatically move the input to the next line. You might alternately use a function that reads characters from the input until a new line is encountered to skip over things you don't need to read. The latter has the advantage that you would not need a place to store things you don't care about.
- If reading into a `double` variable with `scanf` or `fscanf`, you will need the `%lf` format specifier (a "long float") so it knows you have the larger memory chunk available to write as a `double` rather than a `float`. The compiler should warn you of this if using `-Wall`.
- You can copy one C string to another using the `strcpy` function, defined in `string.h`. If you have strings declared as

  ```
  char str1[256];
  char str2[256];
  ```

  the following would copy the contents of `str1` to `str2`:

  ```
  strcpy(str2, str1);
  ```

- Valid values for latitudes range from -90 (southernmost) to 90 (northernmost) and for longitudes range from -180 (westernmost) to 180 (easternmost).
- If you were doing this in an object-oriented language like Java or C++, you might create an object to represent each waypoint and provide methods to compare them, print them, *etc.*. But here, you should use a `struct` that has fields for a string (array of `char`) and two `doubles` to represent the current waypoint you are considering. Follow the model of the `struct ratio` from earlier, and have functions to operate on your waypoints, place them in their own C source file `waypoint.c`, and have a header file `waypoint.h` with appropriate information (`struct` definitions and function prototypes) that you can include in `extremes.c`.
- This will allow you to store a pointer to a waypoint structure to keep track of the "most extreme" leader in each direction you've encountered so far. Do this rather than keeping track of leaders by remembering indices into an array, so you will gain some extra experience with pointers in C.
- You don't actually need to store all of the waypoints. But, for a little extra practice with arrays and pointers, and in recognition that they might be useful if we were to expand this program to have some additional functionality, let's do this extra step. Store pointers to all of the waypoints in an array as they're being read in, then traverse that array to find the answers. You are given the number of waypoints at the top of the file, so you can dynamically allocate enough space for it before you start reading and creating waypoints. At the end, also perform a search for each "winner" waypoint in the array to print out its index along with the other information about the waypoint. This could alternately be accomplished by storing the waypoint number in the structure, but you should not do that.

Submission

Commit and push! And list your commits to be considered for the development process evaluation in the table in your repository's `README.md`.

Grading

This assignment will be graded out of 60 points.

Feature	Value	Score
Command-line parameter	2	
Read input file	2	
`waypoint` structure and functionality	15	
Array of `waypoint` pointers created and filled	5	
Correctly find extreme waypoints	10	
Print all waypoint information for extreme winners	4	
Search for and print indices of winners	4	
All cleanup done properly (closing files, freeing memory)	4	
Documentation	10	
Design/style/efficiency	3	
`Makefile`	1	
Total	60	