UDACITY

Return to Classroom

Build an OpenStreetMap Route Planner

REVIEW

CODE REVIEW 9

HISTORY

Meets Specifications

Dear fellow developer,

I'm thrilled to congratulate you on passing your first project! 🌟
This is a significant milestone in your programming journey, and you should definitely take some time to celebrate your accomplishment! 👏👏

Your code is clean and well-structured, showcasing your good understanding of programming. However, I've noticed that you could benefit from a deeper understanding of the art of commenting in code. This article presents a detailed discussion on the pros and cons of comments. And if you want to dive even deeper into best coding practices, Clean Code: A Handbook of Agile Software Craftsmanship by Uncle Bob is an excellent book that I highly recommend. Here's a handy summary of the main points in the book.

Learning Recommendation

To further enhance your coding skills, I suggest getting more acquainted with the C++ standard library. It provides a wide array of useful data structures and functions. For a comprehensive understanding, check here for an overview and here for a detailed list of header files.

Code Review

Please make sure to go through the code review. I've left some specific suggestions that could further improve your code.

Further Resources

For code styling and best practices, here are some resources you might find useful:

1. Google C++ Style Guide
2. Collaborative Collection of C++ Best Practices

To assist in adhering to the Google C++ style, consider installing the cpplint extension in your code editor.

I'd also like to share this step-by-step Knowledge answer on debugging your code. While it is specifically created for a project in the C++ ND, its principles can be generally applied to any C++ project.

Once again, congratulations on this significant achievement! I encourage you to keep up this momentum as you progress further. Please take a moment to rate this review and provide any feedback.

All the best in your future endeavors!

Compiling and Testing

The project code must compile without errors using `cmake` and `make`.

Great job! The project code compiles with no errors.

Resources on CMake and Make

In the beginning, it is usually unclear what those build systems or build system generators are and how they can benefit us.
I want to share with you some resources to help with this.

- Overview of CMake and how it manages cross-platform build processes.
- Introduction to CMake by examples applies to single file and multiple directory projects.

Please note that CMake isn't a build system. It's a *build system generator*. This is why we need to invoke make after running CMake. Running CMake generates Makefiles with the appropriate platform dependencies, and running make uses them.

However, we don't need to write Make files, as CMake does this for us. It is good to understand build systems.

- Introduction to Make and how it uses the Makefile generated by CMake to build the system.
- How to write Makefiles. This is just for your information. It is already automated through CMake.

Code must pass tests that are built with the ./test executable from the build directory of the project. See the project submission instructions for more details on how to run the tests.

> Tip: All Todo items mentioned acrross "main.cpp" and "route_planner.cpp" must be completed.

The four tests are passed, and the result is very satisfactory.

Resources on software testing

The test framework used in this project is the Google C++ test. It is recommended to know about testing now because it is an essential element in software development. I want to share with you the following resource for this purpose.

- Introduction to the concepts of Google C++ tests.
- Examples-based introductory article of using C++ tests.
- Explanation of the Google C++ test from work with a video tutorial.

User Input

After running the project should be able to test the code by giving a few different input values between 0 and 100 for the start x, start y, end x, and end y coordinates of the search, and the project should find a path between the points.

> Tip: The ToDo mentioned in "main.cpp" should be completed.

I tested the code with multiple inputs, and it is robust. Well done! 👍

Note: It is important to handle the out-of-range inputs to ensure your code is robust against all the possible ranges of inputs. We can ask the user to enter a value in the range of [0, 100], but how can we be sure he/she will follow our instructions? This part is not considered in your code, but it is an excellent practice to consider. I added a detailed comment in the main.cpp about this.

The coordinate (0, 0) should roughly correspond with the lower left corner of the map, and (100, 100) with the upper right.

Note that for some inputs, the nodes might be slightly off(within +-10% range) the edges of the map, and this is fine.

The coordinates are working as expected. Great job!

Code Efficiency

Your code does not need to sacrifice comprehension, stability, or robustness for speed. However, you should maintain good and efficient coding practices when writing your functions.

Here are some things to avoid. This is not a complete list, but there are a few examples of inefficiencies.

- Running the exact same calculation repeatedly when you can run it once, store the value and then reuse the value later.
- Loops that run too many times.
- Creating unnecessarily complex data structures when simpler structures work equivalently.
- Unnecessary control flow checks.

Your implementation avoids unnecessary complex structures, repeating loops, and control flows.

Resources on coding practices

- For beginners to write code with high readability.
- More advanced advice from an experienced developer.

Resources for performance optimization

- Writing efficient C++ code with performance-improving features.
- Tips for C and C++ performance improvement code optimization.
- Efficient C++ performance programming techniques.

DOWNLOAD PROJECT

9 CODE REVIEW COMMENTS ❯

RETURN TO PATH