AP Computer Science Final Project - README Template

Instructions:
The first step in creating an excellent APCS final project is to write up a README. At this stage, this README file acts as your **project proposal**. Once you've filled in all components, Shelby will read through it and suggest edits. Ultimately, you need a document that adequately describes your project idea and **we must agree on this plan**.

Have one member of your group **make a copy of this Google Doc**. Then, they should share it with all other members **and with Mr. Shelby** so that every group member has edit permissions, and Shelby can add comments on your ideas.

There's a lot of parts of this document that you might not have full answers for yet. Because you haven't written the program yet, it's difficult to think about the **instructions** or **which group members will do which parts**. Even though this is hard to think about, you must have something in these sections that acts as your current plan. However, during the course of the project, you'll **continuously update this document**. This means that you will not be *held* to exactly what you put here - components of this document can change (and it's pretty common!).

There is one exception: the **Features List** section. Once Shelby OKs your README, the Features List section **cannot be modified**. For this reason, it is most important that you get a solid idea of what you want to make and the primary features it will have *now*.

Talk with your group. Consider drawing some pictures of what you think your project might look like. Be precise. When you're ready, fill this out together. Each component in brackets below ([these things]) should be replaced with your ideas. Note that there are several sample READMEs posted on this assignment for you to use as guidance.

------------------When README is finalized, remove everything above this line-------------------

[Project Title]

Authors: Mahanth Mohan, Neel Sudhakaran, Raghab Baruah
Revision: 04/15/2022

<u>**Introduction**</u>:
[In a few paragraphs totaling about ½ page, introduce the high-level concept of your program. What this looks like depends a lot on what type of thing you are making. An introduction for an *application* will look different than one for a *game*. In general, your introduction should address questions like these:
What does your program do?

Our program is a Visual Studio Code like code editor that allows the user to code intelligently through the help of contextual suggestions, based on the current code context. This is done through the help of an algorithm that matches commonly used Java code snippets sourced from GitHub, in the form of JSON files that are stored on the user's permanent storage, and displays a suggestion menu with a selected range of n "most-relevant" code snippets for autocompletion, depending on configuration. Furthermore, it includes other features for enhancing the logical correctness of code. For instance, IndexOutOfBoundsExceptions can be identified before runtime, such that most code checked by the editor has a high guarantee of avoiding runtime exceptions caused by invalid operations. In addition, code that is repetitive or actively causes significant wastage of time and space can be avoided through checks while it is being saved by the user. This helps solve the problem of a developer spending too much time writing code, or debugging it for runtime errors like Exceptions and illegal accesses, and improves logical correctness of written code before runtime. The intelligent suggestions provided by the autocompletion engine will allow the programmer to focus more on the core logic of the program, than retyping commonly used code snippets or maintaining styling guidelines. We are creating this for people like us that code and get annoyed at errors that occur at runtime that could easily be caught when we save our code. The end goal for the code is for it to be an easy to use way to make coding projects easier by optimizing and helping you remove errors from your code. We will have auto completion based on common code snippets as well as checkers for possible errors. Additionally we will have some quality of life to improve readability, as well as efficiency suggestions.

Instructions:
[Explain how to use the program. This needs to be **specific**:
Which keyboard keys will do what?
Here is the list of following keys and their associated functions:

- <Tab> key: Allows the user to autocomplete suggestions based on the determined code context
- <Ctrl-S>: Save the file, running code linting and initiating correctness checks and fixes. If any, they are displayed as an error message near the line that is the source of the error
- <Ctrl-K> + F: Open the Folder/Directory that contains the source code
- <Ctrl-K> + O: Open a single source file
- <Ctrl+X>: Copy to clipboard and delete all copied content
- <Ctrl+A>: Copy all code to clipboard
- <Ctrl+R>: Compile and run program with main method
- <Ctrl+B>: Only build/compile the program to generate bytecode in the bin directory

Where will you need to click?
Will you have menus that need to be navigated? What will they look like?
Do actions need to be taken in a certain order?]

Features List (THE ONLY SECTION THAT CANNOT CHANGE LATER):
Must-have Features:

[These are features that we agree you will *definitely* have by the project due date. A good final project would have all of these completed. At least 5 are required. Each feature should be fully described (at least a few full sentences for each)]

- Text-Editing Widget
- Run/Compile Code
- Save/Open Code to/from A File
- See how common a (manually given) code suggestion is on GitHub
- Sort a list of (manually given) suggestions

Want-to-have Features:
[These are features that you would like to have by the project due date, but you're unsure whether you'll hit all of them. A good final project would have perhaps half of these completed. At least 5 are required. Again, fully describe each.]

- Syntax Highlighting
- Compiler Errors/Warnings
- Inferred Code Suggestions (as opposed to a manual one)
- Integrated Terminal/Console
- Settings

Stretch Features:
[These are features that we agree a *fully complete version of this program would have, but that you probably will not have time to implement*. A good final project does not necessarily need to have any of these completed at all. At least 3 are required. Again, fully describe each.]

- Time Complexity Analysis
- Space Complexity Analysis
- Multi-device Code Editing

Class List:
[This section lists the Java classes that make up the program and very briefly describes what each represents. It's totally fine to put this section in list format and not to use full sentences.]

- Main - Has the main method, starts the event loop
- GitScraper - Gets the necessary information from github (Number of commits for a particular code suggestion)
- SnippetSorter - sorts the suggestions in order of what's most popular
- EditorWindow - Is the window that deals with the editing of source files

Credits:
[Gives credit for project components. This includes both internal credit (your group members) and external credit (other people, websites, libraries). To do this:

- List the group members and describe how each member contributed to the completion of the final program. This could be classes written, art assets created, leadership/organizational skills exercises, or other tasks. Initially, this is *how you plan on splitting the work*.

- Give credit to all outside resources used. This includes downloaded images or sounds, external java libraries, parent/tutor/student coding help, etc.]