COM S 440/540 Project part 0

Start the compiler

1 Overview

1. Create a project at `https://git.ece.iastate.edu`.
   - Set the "Project Visibility" to "Private".
   - Add the Instructor and TA (usernames `asminer` and `rdjiles`) as "Developers", with an expiration date of June 30, 2023.
   - You will use the same git repository for all parts of the project, for the entire semester.
   - You should use the directory structure shown in Table 1.
2. Add all necessary source files, including the `Makefile` and `README.txt`/`README.md` file, to the git repository. **DO NOT** add any files that are automatically generated from source files.
3. Ensure that all changes are committed and pushed to the server.
4. Submit in Canvas.

2 The compiler executable

You must implement your compiler in C, C++, Java, or Python (or a combination), and your `Makefile` should produce an executable (or jar file for Java, or nothing at all for Python) with name "mycc" or something very similar. This will be a command-line compiler, so all interaction will be done using arguments and switches on the command line. All project specs are written assuming you have an executable that can be run directly; in practice you might run "`java -jar mycc.jar` [arguments]" instead of "`./mycc` [arguments]" on the command line. The compiler should *never* prompt the user for anything. Any messages (warnings, errors, or information) not related to the required output should be written to standard error. You may abort execution, or valiantly attempt to continue, after the first error message.

The compiler should be invoked using the following command-line arguments and switches, discussed below.

```
mycc -mode infile
```

As a special case, though, if the compiler is invoked with no arguments at all, you should display (to standard error) usage information and a brief summary of what features are implemented. You may implement additional switches if you like; for example, you might add switches to display information to help you with debugging.

2.1 The mode switch

The mode is an integer from 0 to 9 and corresponds to each part of the project (there will be fewer than 9 parts). We will build up the compiler in pieces, each one roughly corresponding to one of the compiler phases. This allows students to work ahead and not worry about disturbing the graders (e.g., you can work on part 3 while we are grading part 2, because we will test with `-2` and you will test with `-3`). Equally importantly, it allows for regression testing: because part 3 will build upon part 2, if a test input file fails for part 3, you might want to make sure that your part 2 solution can handle it.

Name	Purpose
`Documentation/`	for LaTeX sources of project documentation
`Grading/`	for Instructor and TA feedback (put a short README file inside)
`Source/`	for your source code, which may be further subdivided as you wish

Table 1: Directory hierarchy to use within your git repository.

2.2 Input files

For most modes, it is an error if no input files are specified; your compiler should display an appropriate error message (to standard error, of course) in this case. Your compiler will be tested with at most one input file at a time. Students may, if they wish, implement a compiler that accepts and processes more than one input file at a time.

2.3 Compiler output

Each part of the project will require a different form of output, in a specified format, either to standard output or to a specific output file whose name will be based on the input file name but with a different "extension". Output will always be ASCII text.

2.4 Error messages

All error messages for all parts of the project should be written to **standard error**. When errors are caused by the input file (e.g., a syntax error), the error message should display the filename, line number, and text (as appropriate) that caused the error. This will be specified further in later parts of the project.

2.5 Some assumptions

You may safely assume:

- When grading part n of the project, your compiler will be invoked with first argument "-n", and no other modes will be specified on the command line.
- Input file names will appear last (after the mode) and will never begin with "`-`" or contain spaces. They could, however, contain directory separators or several periods.

3 Requirements for part 0

- If no arguments are given, display usage information to standard error.
- For mode 0, ignore any input files (you may give a warning about this) and display, to standard output, the following information (this will help with grading):
  - The compiler name
  - Author and contact information
  - A version number and a date of "release"

4 Examples

Running

```
mycc
```

should display something like the following on standard error.

```
Usage:
       mycc -mode infile

Valid modes:
        -0: Version information only
        -1: Part 1 (not yet implemented)
```

Running

```
mycc -0
```

should produce output like the following (to standard output).

```
My bare-bones C compiler (for COM 440/540)
       Written by Andrew Miner (asminer@iastate.edu)
       Version 2.1, released 5 January 2023
```

5 Grading

Points		**Description**
10		**Documentation**
	5	`README.txt` or `README.md`
		Explains how to build and run the compiler, and how to build the documentation.
	5	`developers.pdf`
		Built from the LaTeX source `developers.tex`. Add a new section to this document for each part of the project. This should explain, to other developers (students in the course), the purpose of each source file, the main data structures used, and give a high-level overview of how your code works. (Note: this is really overkill for this part of the project; I do not expect much depth here yet.)
10		**Ease of building**
		How easy was it for the graders to build and run your compiler based on your `README` file. If your source code must be compiled (i.e., you are not using Python), then there should be a `Makefile` and running `make` should build everything.
10		**Working executable (test on `pyrite.cs.iastate.edu`)**
	5	Correct behavior for `-0`
		Version and author information is written to standard output.
	5	Correct behavior for no arguments
		Usage information, and summary of what is implemented, written to standard error.
30		**Total for students in 440**
30		**Total for students in 540**

6 Submission in Canvas

In Canvas, submit the URLs to clone your git repository via ssh and https. We will grade your work by cloning a copy of your repository on `pyrite.cs.iastate.edu`, and testing there. Students are strongly encouraged to test on `pyrite` themselves.