COM S 440/540 Project part 0

Start the compiler

1 Overview

1. Create a project at `https://git.linux.iastate.edu` (you will need to create an account if you do not have one).
   - Set the "Project Visibility" to "Private".
   - Add the Instructor and TA (usernames `asminer` and `lliu`) with roles of "Reporter".
   - You will use the same git repository for all parts of the project, for the entire semester.
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 or C++ (or a combination), and your `Makefile` should produce an executable named "mycc". This will be a command-line compiler, so all interaction will be done using arguments and switches 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 [-o outfile] [infile] ...  [infile]
```

The "`[]`" notation indicates that an item is optional. As a special case, though, if the compiler is invoked with no arguments at all, you should display usage information to standard error. If you choose, you may implement a more permissive compiler (e.g., maybe you allow the mode or output file switches to appear in any order). 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 5, and corresponds to each part of the project. 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.

2.2 The output file

Each part of the project will require a different form of output, in a specified format. If no output file is specified, then the output should be written to standard output (this is why errors should go to standard error). Otherwise, output should be written to the specified output file.

2.3 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 output, of course) in this case. Generally, you will need to process a single input file. There may be an extra credit option to process more than one input file.

2.4 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.
- If an output file is specified, it will appear directly after the mode.
- Output file names, if specified, will never begin with "`-`".
- Input file names will appear last (after the mode and after the output file name, if present) and will never begin with "`-`".

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 the output stream, 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 [options] infile

Valid modes:
        -0: Version information
        -1: Part 1 (not implemented yet)
        -2: Part 2 (not implemented yet)
        -3: Part 3 (not implemented yet)
        -4: Part 4 (not implemented yet)
        -5: Part 5 (not implemented yet)
```

```
Valid options:
        -o outfile: write to outfile instead of standard output
```

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 0.1
       5 January, 2021
```

Running

```
mycc -0 -o out.txt
```

should produce the same output in file `out.txt`.

5 Grading

Points	Description
15	**Documentation**
5	`README.txt or README.md` Explains how to build the compiler and documentation. For each part of the project, explains which features are implemented.
10	`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 your compiler and documentation from the `README` file. For full credit, simply running "`make`" should build both the documentation and the compiler executable, and running "`make clean`" should remove all generated files.
15	**Working executable (test on `pyrite.cs.iastate.edu`)**
5	Correct behavior for `-0` Version and author information is written to output.
5	The `-o` switch works Using `-o` causes output to go to the specified file.
5	Correct behavior for no arguments Usage information written to standard error.
40	**Total for students in 440**
40	**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.