Objectives

To investigate appropriate tools to assist with the implementation of the parsing phase of the compiler. To participate in the peer evaluation activity to assess and to aid with the progress of the parser development. To develop fully documented and tested parser for the chosen language implemented in the programming language of the developer’s choosing.

Reading Assignment

If you have not done so already, please read the “Mastering Markdown” and “Documenting Your Projects on GitHub” GitHub Guides available at https://guides.github.com/. To have a better understanding of the parser implementation of the compiler, you should also read Chapter 3 in the BCD course textbook.

Syntax Analysis

In this lab you will expand the front-end of your compiler by implementing a parser for it. For this task, you will use a syntax tool appropriate for the source programming language of your choosing. Please note that it is likely that you will need to expand and/or modify parts of your scanner to get your parsing to work correctly. - this is okay and expected!

Grammar and Parsing Tool

Before you can begin the implementation of this portion of the compiler’s front-end, you need to locate the grammar specification for your chosen language. You should also decide if you need to or want to extend the grammar in any way. Next, you need to explore and choose an appropriate parsing tool to use. For example, if you used JFlex as your scanner generator you can use parsing tools such as CUP or BYacc/J that utilize LALR algorithm or ANTLR that is based on LL implementation.

Once you have selected the parsing tool, your task is to add the rules for each of your language’s grammar features using your chosen parsing tool. You do not need the actions to build the parse tree yet. In fact, it may help to first concentrate on getting the rules written and conflicts resolved before you add actions. Your parser should accept the grammar as specified for your chosen source language, but you can rearrange the productions as needed to resolve conflicts. Some conflicts can be resolved in a multitude of ways (rewriting the productions, setting precedence, etc.) and you are free to take whatever approach appeals to you. All you need to ensure is that you end up with an equivalent grammar. All conflicts and errors should be eliminated at this stage.

Parse Tree

Your parser should generate a parse tree. Most parsers will require you to write grammar rules and actions for these rules. You could add actions for each rule as you go, but I recommend that you wait until all rules are debugged (previous step) and then go back and add actions. The action for each rule will be to construct the section of the parse tree corresponding to the rule.
Remember that syntax analysis is only responsible for verifying that the sequence of tokens forms a valid sentence given the definition of the grammar. Given that some grammars can be somewhat loose, some apparently nonsensical constructions will parse correctly and, of course, you are not yet doing any of the work for verify semantic validity (typechecking, declare before use, etc.).

Once you have the full grammar from your source language specification operational, take a chance to personalize it to handle your common mistakes! Which keywords do you often misspell, what particular aspects of your source language might trip you up, how would you like these to be handled?

Pick out some simple errors that you think you can tackle and add some rudimentary error handling capabilities into your own parser. In your README, describe which errors you attempt to catch and provide some sample instances that illustrate the errors from which your parser can recover. To receive full credit, your parser must do some error recovery, but my expectations for the standard requirements are quite modest. Simple recovery, say, at the statement and declaration levels, is enough. I just want you to explore and experiment with error handling. Feel free to attempt more ambitious error handling.

**Required Deliverables**

I expect your programs to accept exactly the reference grammar with the added constructs, have no parser conflicts, and have some rudimentary error recovery. Most points are allocated for correctness, with the remainder reserved for the writeup of your error handling explorations in your README. Specifically, this assignment invites you to submit, using GitHub, the following deliverables. You are to use the same repository from your previous lab as it is to contain the implementation of your full compiler at the end of the semester.

- Fully documented programs with the parser’s implementation that are free of errors. Your program(s) should be stored in the `src/` directory of your repository. It should be clear from your directory structure and file naming conventions what constitutes your parser.

- A comprehensive documentation on the description of the tools and languages necessary for your implementations and the detailed steps to follow to build/run your parser. This information should be written in the Markdown language and be added to your repository’s `README` file.

- Small test files used to test your parser. You must include at least five test program(s) that are stored in the `tests/` directory of your repository. Any sort of input is fair game and you will want to be sure yours can handle anything that comes its way, correctly parsing it if possible or reporting reasonable errors if not.

**Peer Evaluation**

During the lab session on March 5 we will have a peer review session. The goal of this session is to assess overall design of your parser, to catch any big red flags, and to aid with the completion of your implementation. During this session each developer will be partnered with a peer developer to explain your overall parser design and to complete a code walkthrough of your selected completed implementation. Each reviewer will complete a peer review summary, which will be provided to the class beforehand.

Before the peer evaluation session please ensure that you have made a tremendous progress towards your implementation. You must have some code by this time that can be reviewed. It is
not necessary to have completed test cases and a fully implemented parser by the peer evaluation session.

Recommended Lab Completion Timeline

The following timeline is recommended to ensure you are able to complete this lab by the due date.

- Week 1 (Feb 19 - Feb 26): Research and finalize the decisions regarding the parsing tool. Locate and study the source language’s grammar. Prepare for the implementation.
- Week 2 (Feb 26 - March 5): Begin implementation. You should be at least half way finished before the peer review session.
- Week 3 (March 5 - March 12): Complete implementation and testing.

Adhering to the Honor Code

In adherence to the Honor Code, students should complete this assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else’s work. Additionally, since there are many compiler implementations available online, please mention any outside tools and projects you have used to help you. Deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code. Please see the course instructor if you have questions about this policy.