Objectives

To explore design decisions needed to be considered before compiler development work begins. To investigate appropriate tools to assist with the implementation of the scanning phase of the compiler. To participate in the peer evaluation activity to assess and to aid with the progress of the scanner development. To develop fully documented and tested scanner 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 scanner implementation of the compiler, you should also read Chapters 2.1-2.9 in the BCD course textbook.

Accessing the Laboratory Assignment on GitHub

To access the laboratory assignment, you should go into the #labs channel in our Slack team and find the announcement that provides a link for it. Follow the link to accept the laboratory assignment and have a new GitHub repository created for you to access the assignment’s starting materials and to store the completed version of your assignment. Now, you should clone the repository and begin your work. Unless you provide the instructor with documentation of the extenuating circumstances that you are facing, not accepting the assignment means that you automatically receive a failing grade for it.

Design Decisions

Before you can begin the implementation of this portion of the compiler’s front-end, you need to finalize several decisions.

1. **Source Language:** Following the insights you gathered during lab0 you must now decide on the source language that your compiler will use to translate. This is a crucial decision as the implementation for the scanner and the parser that you complete in this and subsequent labs will depend on the source language that you choose. As you decide on your “toy” source language please consider the features that it supports and the comprehensive description of the language that is available to you. For example, your source language should support key programming features and provide specifications for the following:

   - reserved keywords,
   - identifiers and constants,
   - variables,
   - data types,
   - functions,
2. **Implementation Language:** Next, you need to decide on the programming language that you will use to develop your compiler. For this decision, there are two factors you should consider: (1) your expertise in a specific language or ability to learn it; (2) the availability of automated tools for compiler development, such as lexical and parser generators. For example, there are a number of lexical analyzer generators available for various languages, such as **JFlex** for Java, **ocamllex** for OCaml, **Alex**, **ruby-lex** for Ruby, **jison** for JavaScript, **ply** for Python. Most of these generators are derived from **lex** (or **flex**), the original lexical analyzer generator for C. Spend some time investigating most recently supported generators and choose a programming language that has well-documented and well-supported generators available for use. If you need any tools installed on Alden machines, please let the instructor know. You are also welcome to use your own machine or borrow a **raspberry pi** computer.

**Lexical Analysis**

In this lab, you will start with the first step of implementing a compiler by developing the lexical analyzer (scanner). For this first task of the compiler’s front-end, you will use a lexical generator tool appropriate for the source programming language of your choosing. Your scanner will transform the source file into a series of meaningful tokens containing information that will be used by the other stages of the compiler. You should begin by making a list of all items your scanner needs to handle based on the features of your source language.

At the minimum, your scanner should be able to:

- skip over white space (if appropriate for your source language),
- recognize all keywords and return the correct token,
- recognize punctuation and single character operators and return the correct token,
- recognize supported data types and constants and return the correct token,
- recognize identifiers and return the correct token,
- report lexical errors for invalid characters and any other improper lexical notations you are able to catch at this point.

It is recommended that you add token types one at a time and test after each addition. For any items that are not explicitly defined in your source language documentation, you can make any assumptions as appropriate.

**Required Deliverables**

This assignment invites you to submit, using GitHub, the following deliverables:

- Fully documented programs with the scanner’s implementation that are free of errors. Your program(s) should be stored in the src/ directory of your repository.
• 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 scanner. This documentation should be written in the Markdown language inside your repository’s README file. Please customize the existing README file for this purpose.

• Small test input files written in your source language that are used to test your scanner. You must include at least three 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 tokenizing it if possible or reporting some reasonable error if not.

Peer Evaluation

During the lab session on February 12 we will have a peer review session. The goal of this session is to assess overall design of your scanner, 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 scanner 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 finalized your design decisions and 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 scanner 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 (Jan 29 - Feb 5): Research and finalize the design decisions. Prepare for the implementation.

• Week 2 (Feb 5 - Feb 12): Begin implementation. You should be at least half way finished before the peer review session.

• Week 3 (Feb 12 - Feb 19): 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.