COMP 520 Compiler Design
Individual Assignment #1

Scanning and Parsing MiniLang

Due: Saturday, January 26, 11:59 PM

Overview

In class, we focus primarily on the theoretical side of compilers, with only a brief overview of the tools themselves. The purpose of this first assignment is to get you up to speed with the practical side of scanning and parsing, and to introduce you to modern compiler tools. You may use any toolchain you wish (in class we show both flex/bison in C and SableCC in Java) provided that it runs on the SOCS Trottier machines. You may also hand-code a scanner and/or parser if you're interested! Use this time to explore the various options and find a set of tools that you will be comfortable with for your course project.

Question 0: *Name the COMP 520 dragon*

Suggest a name for the dragon on the COMP 520 home page: `http://www.cs.mcgill.ca/~cs520/2019/`. The winning name gets bonus points!

Question 1: *Example Programs* (10 points)

Write the following example programs ending with file extension `.min`

(a) **Syntactically-Correct Example Programs** (5 points)

Write two syntactically correct `MiniLang` programs that perform an *interesting* computation using a variety of language features (no Fibonacci or factorial!).

(b) **Syntactically-Incorrect Example Programs** (5 points)

Write five incorrect `MiniLang` programs which each exhibit a *different* scanning or parsing error. Ideally, each program should be minimally sufficient to trigger the error, so think carefully about your program. Include a comment at the start of each file describing the intended issue.

Note that although we only require 7 small example programs for this question, you should prepare a more substantial test bank for debugging your submission. As part of evaluating your work, we will execute our own comprehensive test suite that covers all language features.

Question 2: *Scanner/Parser for MiniLang* (20 points)

Implement the scanner and parser for `MiniLang` using your chosen toolchain. The core language specification was defined by Vincent Foley and augmented with input from our class discussion. A formal overview can be found at `http://www.cs.mcgill.ca/~cs520/2019/assignments/Assignment1_Specifications.pdf`. Note that this may change as we discover the intricacies of language design - report any issues/challenges you encounter and we will discuss in class!

For this first assignment, your compiler must support 3 modes as command-line arguments:

- `scan`: Outputs `OK` if the input is lexically correct, or an appropriate error message
- `tokens`: Outputs the token types, one per line, until the end of file. Tokens with associated data (literals, identifiers, etc) should be printed with their respective information
- `parse`: Outputs `OK` if the input is syntactically correct, or an appropriate error message

Normal output (`OK`, tokens) must be sent to `stdout` and your compiler must exit with status code 0. If an error occurs, the message must be displayed on `stderr` and your compiler must exit immediately with status code 1. Error messages should be descriptive (look into error reporting in your toolchain) and formatted as "`Error: <description>`".

To organize your submission, please follow the starter code and directory structure provided on the course GitHub repository: `https://github.com/comp520/Assignment-Template`. As part of the template, we provide 2 scripts that you must modify to support your assignment particulars:

- `build.sh`: Builds your compiler using `Make` or similar
- `run.sh`: Takes two arguments (mode and input file) and executes your compiler binary

Comments found in both files provide the exact requirements. Since a large portion of grading is performed automatically, please follow the input/output specifications *exactly*. You must be able to run the provided scripts on the SOCS Trottier machines. A convenience `test.sh` script executes all programs in the `programs` directories using the above scripts and reports any issues found.

Submission

On myCourses, hand in a tar.gz file of the form `firstname_lastname.tar.gz` (all lowercase, please use .tar.gz). The structure of files inside that tarball should be:

```
firstname_lastname/
    README   (Name, student ID, any special directions for the TAs)
    DragonName.txt (Answer to Question 0)
    programs/
        1-scan+parse/
            valid/   (Two correct programs)
            invalid/ (Five syntactically incorrect programs)
    src/     (Source code and build files)
    build.sh (Updated build script)
    run.sh   (Updated run script)
```