COMP 520 Compiler Design
Group Milestone #1

Scanner, Parser and Pretty Printer for GoLite

Due: Friday, February 22, 11:59 PM

The first milestone of the `GoLite` project covers the lexical, syntactic, and intermediate representation of your compiler. Just like your assignments, you may use any toolchain you wish (in class we showed 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 feel ambitious!

Question 1: *Example Programs* (10 points)

Design a test suite for your compiler, writing the following programs with extension `.go`

1. **Syntactically-Correct Example Programs** (5 points)

   2 syntactically correct `GoLite` programs per team member that perform an *interesting* computation (i.e. either 4 or 6 programs, depending on the number of people in your team). Try to use a variety of language features!

2. **Syntactically-Incorrect Example Programs** (5 points)

   10 incorrect `GoLite` programs per team member that exhibit a *different* scanning or parsing error (i.e. either 20 or 30 programs, depending on the number of people in your team). Ideally each program should be minimally sufficient to trigger the error, so think carefully about your programs. Include a comment at the start of each file describing the intended issue.

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

Question 2: *Scanner, Parser and Pretty Printer* (30 points)

Implement the scanner, parser, AST, and pretty printer for `GoLite` using your chosen toolchain. The language specification was defined by Vincent Foley and is available at `www.cs.mcgill.ca/~cs520/2019/project/Milestone1_Specification.pdf`. As noted in the document, some weeding passes are also required as part of this milestone.

For this first milestone, 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, pretty printer) 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 (i.e. `./run.sh <mode> <filename>`)

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.

Question 3: *Design Decisions and Team Work* (10 points)

Briefly discuss the design decisions you took in the design and implementation of your scanner, parser, AST, and pretty-printer. If there are parsing issues that you implemented as a weeding phase, document them here. Also include in this discussion

- The rationale of the implementation tools and language that you chose; and
- Summarize how your team is organized and what each team member contributed.

Note: You should keep notes on each phase, as this will help you generate the final project report.

Submission

Create a tag in your Github repository named *milestone1* (lowercase, no extra characters). Information about creating git tags can be found at: `http://git-scm.com/book/en/v2/Git-Basics-Tagging`. Your project should be kept in the following format

```
/
    README   (Names, student IDs, any special directions for the TAs)
    programs/
        1-scan+parse/
            valid/   (correct programs)
            invalid/ (incorrect programs)
    doc/     (Design documents)
        milestone1.pdf
    src/     (Source code and build files)
    build.sh (Updated build script)
    run.sh   (Updated run script)
```