EGRE591 Compiler Project
Part 1 - Lexical Analysis
Spring 2024

The semester project involves writing a compiler for the language **Toy C** (toyc), a subset of C. Source code files for toyc programs should end in extension '`.tc`', e.g. `helloWorld.tc` .

1 Lexical Specification of `toyc`

1. The language is case-sensitive.

2. Keywords:

```
int      char     return   if       else     for
do       while    switch   case     default  write
read     continue break    newline
```

   Note that keywords must be lowercase.

3. Text following `//` up to the end of a line is ignored by the scanner, as is all text delimited by `/*` and `*/`. Comments may appear after any token and can be nested. A "run-away" comment (i.e. one that is not closed before the end-of-file is reached) should produce an error message.

4. White space between tokens is optional (and ignored) except where required to allow the scanner to discern between tokens.

5. An identifier (token **ID**[1]) is a letter followed by letters or digits:

$$\begin{array}{rcl} \textbf{letter} & \rightarrow & \texttt{[a-zA-Z]} \\ \textbf{digit} & \rightarrow & \texttt{[0-9]} \\ \textbf{ID} & \rightarrow & \textbf{letter} \; (\; \textbf{letter} \mid \textbf{digit} \;)^* \end{array}$$

6. Token **NUMBER** matches unsigned integers and reals.

$$\begin{array}{rcl} \textbf{digits} & \rightarrow & \textbf{digit digit}^* \\ \textbf{optional\_fraction} & \rightarrow & (\; \texttt{.} \; \textbf{digits} \;) \mid \epsilon \\ \textbf{optional\_exponent} & \rightarrow & (\; \texttt{E} \; (\texttt{+} \mid \texttt{-} \mid \epsilon \;) \; \textbf{digits} \;) \mid \epsilon \\ \textbf{NUMBER} & \rightarrow & \textbf{digits optional\_fraction optional\_exponent} \end{array}$$

7. Character literals (token **CHARLITERAL**) are delimited by single quotes (`'`) and cannot contain newlines. An unterminated char literal should be reported as an error. Empty char literals (i.e. `''`) are allowed.

8. String literals (token **STRING**) are delimited by double quotes (`"`) and cannot contain newlines. An unterminated string should be reported as an error. Empty string literals (i.e. `""`) are allowed.

[1]Token names are represented by all caps. You must use the exact name as specified.

9. The relational operators (token **RELOP**) are: `==`, `!=`, `<`, `<=`, `>=`, and `>`. Note that `!=` denotes $\neq$ (not equal).
10. The **addop**'s (token **ADDOP**) are `+`, `-`, and `||`.
11. The **mulop**'s (token **MULOP**) are `*`, `/`, `%`, and `&&`.
12. The lexeme for token **ASSIGNOP** is `=` .
13. Other tokens: **LPAREN** (`(`), **RPAREN** (`)`), **LCURLY** ({), **RCURLY** (}), **LBRACKET** (`[`), **RBRACKET** (`]`), **COMMA** (`,`), **SEMICOLON** (`;`), **NOT** (`!`), and **COLON** (`:`).
14. Illegal characters should be ignored with the printing of an appropriate warning message to standard output.
15. For part1 all errors (not warnings!) should be fatal, i.e. you should print an error message to standard output and exit the program.

2 Program Development

You may choose any program language you wish for your project according to the specifications in part0. However you must completely implement your state machine 'by hand', i.e. you cannot use any language features or libraries that automatically generate a DFA (state machine) from regular expressions. In addition, the customs and practices for software development are somewhat different for different languages; therefore the directory structures, files used, and the way your project will be organized will vary depending on which language you choose. Unless you have considerable prior experience in organizing and managing a project of this scope, it is strongly recommended that you follow the customs and practices of other experienced programmers for organizing large projects.

3 Command Line Arguments

The only string required on the command line when executing your program is the name of the input file. Your program must also provide a certain level of control from the command-line. For part1 of the project you should provide actions for the `help`, `debug`, and `verbose` flags in the manner stated in this example below.

```
liberty:~/compilers/toyc/> java -cp build/classes:../interfaces.jar parser.tc -help
Usage: java [classpath] parser.tc [options] toyc_source_file

where options include:
   -help           display this usage message
   -debug <level>  display messages that aid in tracing the
                   compilation process. If level is:
                      0 - all messages
                      1 - scanner messages only
   -verbose        display all information

liberty:~/compilers/toyc/>
```

Note that you will be adding legal command line options as your compiler is developed stage by stage. The default condition of your scanner should be with no debug flags set (on) and in non-verbose mode.

4 Example Run

Date file `tests/scanTest.tc`:

```
hello char int while <= !=
123 "hello" /*
* / && /* ||
> */ */ *
// this is a comment
```

Run of scanner (note that '`debug 1`' is turned on):

```
liberty:~/compilers/toyc/> java -cp build/classes:../interfaces.jar parser.tc -debug 1 tests/scanTest.tc
[SCANNER] (<ID>,"hello")
[SCANNER] (<CHAR>,"char")
[SCANNER] (<INT>,"int")
[SCANNER] (<WHILE>,"while")
[SCANNER] (<RELOP>,"<=")
[SCANNER] (<RELOP>,"!=")
[SCANNER] (<NUMBER>,"123")
[SCANNER] (<STRING>,""hello"")
[SCANNER] (<MULOP>,"*")
[SCANNER] (<EOF>,"EOF")
[SCANNER] Total tokens: 10
liberty:~/compilers/toyc/>
```

Your token/lexeme pairs must be reported in exactly the format shown above and use the token names given in the specification.

5 The Part1 Assignment

5.1 State Diagram

You are to draw and turn in (in class, i.e. a hardcopy) a state diagram representing the finite state machine (FSM) for toyc. Your FSM need not represent the key words individually. If you are working with a partner you should work on the FSM together and only turn in a single hardcopy with both of your names on it.

5.2 The Scanner

Your project source code will be submitted via Gradescope as a single zipfile in the same manner as you submitted part0. Assignments are due at the beginning of class on their due date.

Due dates

for state diagram: Wednesday, February 7

for complete scanner: Wednesday, February 21