CS 3323 - Principles of Programming Languages
Spring 2022 - Assignment 1

Download the files pl-scanner.yy, driver.cc, tokens.h and Makefile from the assignment directory space. Then, perform the necessary modifications to the rule's file to recognize the below strings and tokens. The token codes can be found in the tokens.h header file.

The assignment is due on Monday February 21, 11:59pm CT. Files must be uploaded by then. Late policy deduction applies.

You can work in groups of up to 3 students. Groups remain the same for the whole semester. **All students must upload the corresponding files.**

1. (1.5pt) Add the necessary rules to recognize the arithmetic operators: $+, -, *, /, +=, ++, <=, >=, ==, \sim=, <, >$. See the file tokens.h to determine the constants to be used (they start with the prefix 'OP_').

2. (2pt) Modify the rule returning the T_ID token to recognize all identifiers matching the following:

   - identifiers **must start** with the character '@'.
   - the character immediately following the '@' **must be a letter**.
   - letters used in identifiers can only be in lower case, that is, letters 'a' through 'z' are valid, while 'A' through 'Z' are not.
   - characters following the initial '@' and the first letter **can be** any letter, digit or the underscore '_' symbol.

   Examples of valid identifiers are (commas used separate identifiers): @iter, @iii, @a1i, @a_xe, @b_p, @b20, @a_b_c_2_.

   Some examples of invalid identifiers are (commas used separate identifiers): __, _2, at, 2_, 12b_5, abC24, @_b43, _delta, @, @Ijk.

3. (2.5pt) Create a new rule to recognize floating point numbers. The rule should return the token L_FLOAT:

   - Can start with a digit, the '+' or a '-'
   - Both the integer and fractional part should consist of at least one digit
   - The integer and fractional part should be separated by a '.'

   Examples of acceptable floating point numbers are: "10.0", "+1.5", "-10.50000", "0.9".

   Examples of strings that should be rejected are: "0.", ".01"

4. (1.5pt) Create a rule to recognize the following keywords: INTEGER, FLOAT, FOREACH, BEGIN, END, REPEAT, UNTIL, WHILE, DECLARE, IF, THEN, PRINT. The tokens corresponding to the keywords are those with the prefix 'K_' in the tokens.h header file.

For convenience, a Makefile is provided, but you are not required to use it. Run:

```
make
```

to rebuild the scanner generator (lex.yy.c), and to recompile the driver.

To validate your scanner's functionality, prepare a file containing a set of valid and invalid inputs. It's recommend to test the scanners with different groups of inputs. For example, test your identifier rule first, then integer numbers, then floating point numbers, then keywords and so on.

You can redirect any of the input test file by doing:

```
./pl-scanner.exe < input.txt
```

Your grade will be determined by comparing the output produced by your scanner against our own set of inputs.

Some recommendations:

- Always include an action to execute even if it's an empty one. i.e. "{ }".
- The action associated to a detected pattern **must start** in the same line as where you write the pattern.
- Remember the issue with pattern prefixes (See slides and or corresponding class recording of Lexical Analysis).
- If something doesn't work, isolate the pattern you want to test in another file and test just that pattern.
- Don't feel afraid to run "flex -h" in the command line and test other flags/options.
- Don't insert any type of line or block comments as part of the regular expression patterns nor the actions. It will likely get Flex (and you) confused.

Several online resources can be found in the web, for instance:

- `https://www.cs.virginia.edu/~cr4bd/flex-manual/`,
- `http://alumni.cs.ucr.edu/~lgao/teaching/flex.html`,
- `http://web.mit.edu/gnu/doc/html/flex_1.html`,
- `https://westes.github.io/flex/manual/`.

More resources can be found by searching for the key terms: C scanner generator.

Do not change the driver file nor the tokens.h file since the actual integer values will be used for grading. Do not print anything to the output.

Every student should **upload a single file named: ABCDEFGHI.yy**, where ABCDEFGHI is the 9-digit code identifying the student (not the 4+4).