University of Toronto
CSC467F Compilers and Interpreters, Fall 2018

Assignment 1 FAQ

Shirley Yang (myshirley.yang@mail.utoronto.ca)

General

Your lexer should reach the following goals.

1. Create separate/different token types for keywords, variable types, function, identifiers, operators. (If you take a look at the parser code, there's no way for bison to recognize keywords if you use just one token type for everything, so separate token types are needed.)

2. Whitespace and comments should be safely ignored. You don't have to generate tokens for comments and whitespaces.

3. Store parsed values (or types if your token types don't cover them) into the "yylval" union. For example, you need to store values for integer/float, identifier name, and dimension of the vector types. ("yylval" union should be initialized inside the "parser.y" file)

Note:

- Unmatched parenthesises are **NOT** handled in Assignment 1, since regular expressions cannot describe it. They will be handled in the next assignment.

- The structure of the "yylval" union types of tokens used are **your own design choice**. However, your lexer should allow your parser compiler backend to **uniquely identify** different keywords, identifier names, int/float values identifier names, etc.

  - Example 1: there should be different tokens types for "while" and "if", "bool" and "int"; also different token types for operators such as "<" and "||"
  - Example 2: "ivec2" and "ivec3" may or may not share the same token type, but vector dimension should be specified in "yylval" if token type is shared. You may:
    - have 1 token type for all functions and save the function names in "yylval", or,
    - create separate token types such as "DP3_FUNC", "LIT_FUNC", etc.

Identifiers

- Identifiers should not start with numbers.

- Length of identifier names cannot exceed 32 bytes.

Comments

- You only need to check /**/ style comments. (Don't need to check for "//. . . ")

- If you found a comment starting with "/*", but no "*/" is recognized till the end of file, an error should be generated.

- You do not need to find matching pairs of "/*" and "*/". After meeting the open comment "/*", treat the closest "*/" as the closing comment.

Numbers

Integer/Floating Point

- Integers should be in range of -32767 to 32767. Floats should be in range of -1E+37 to 1E+37. Otherwise, output errors.
- Floats beginning with no numbers (.345) and numbers (0.345) should be both supported.
- Floating point precision: do not need to generate errors if you cannot represent a floating point number using its full precision (E.g. If you see something like 9.000000000000000000000012345678 in program, just parse it to float.)

Scientific and Other Notations

- The lexer should recognize float point numbers in scientific notations and evaluate/store their appropriate values. The compiler only need to support cases when the exponent is an integer. (E.g. Supports 1E5, 2E6, but not 1E1.5)
- Range checks for float also apply here.
- Suffix f or F is not supported for floating numbers, such as "0.0f" or "1.0F".
- NaN is also not supported.

Octal numbers

In this lab, you do **NOT** need to support octal numbers.

- As a result, if you see octal representation (Some int number starting with "0" except for 0 itself: e.g. 012, 015), you need to `output an error`.
- But if you do want to support octal numbers in your implementation (which is not required), you need to reject incorrect/out of range octal values (Ex. 018 is not a valid octal number; "8" is over the range); and evaluate correct values for octal numbers.

Line Number

- Line numbers need to be tracked. (So when an error occurs, correct line number would be printed out)
- Note: you need to keep track of the line numbers inside the comments as well.

Syntax Error Check

- Lexer should detect ALL lexical errors, but it should NOT check parsing/semantic errors.
- Examples of errors we are looking for specifically:
  - Invalid characters not in the grammar. E.g. %, #, etc.
  - Numbers incorrectly followed by identifier. E.g. 123var.
  - Open comments "/* . . . " without closing