CS335 Assignment - 1

Aditya Tanwar
200057

January 2023

Problem 1

This part explains the files, regular expressions, and assumptions for the `Flex` file written for CSEIITK.

Running the Flex files

The `Flex` file uses `C++` code, which uses the `STL` library.

The bash file used for the first part of the assignment is `P1.sh` in the directory `problem1`.
I/O redirection has been used in the script to read from a file, and write to a CSV file. If two arguments are given to the script, then the first file is used as the input stream, and the output is written to the second file. Otherwise, it is assumed that one argument is given, which is the input file's name. In this case, "`.csv`" is appended to the given file name, and the output is printed in this CSV file.

Thus, the bash file can be run in any of the two following ways:

```
./P1.sh input.335
./P1.sh input.335 output.csv
```

Data Structures Used

Some data structures, specifically `unordered_map` and `vector`, have been used from the `C++ STL` library. A `vector` (`lexemes`) is used to keep track of the order of appearance of lexemes. Two maps are used, one of which tracks the token assigned to a lexeme (`lexeme_token`) and the other which keeps the count of a lexeme (`lexeme_count`).
The `Flex` directive `%option yylineno` has been used to track the line number which is used when reporting errors.

Regular Expressions

- The regular expression for an "Identifier", "Number" (not float) were made simply using the definitions provided in the assignment PDF. The RegEx for numbers allows numbers starting

with (any number of) zeroes.
Further, lookahead with delimiters was used to accept both cases like "`1..10`" and "`1.`". Also, in cases like "`1-2`", three lexemes are scanned, instead of clubbing the `-` sign with `2` to produce a single lexeme.

- For a "String", multiline strings have not been handled currently, since it was said on Piazza that they won't be included in test cases.
- For a "Float", the RegEx accepts both floats using a decimal point and those using exponential form.
- The regular expressions for rest of the tokens are straightforward.

Problem 2

This part explains the files, regular expressions, and assumptions for the `Flex` file written for Java.

Running the Flex files

The C++ library (and Data Structures) used and the bash file, `P2.sh` are similar to the ones used for Problem 1.

Thus, the bash file can be run in any of the two following ways:

```
./P2.sh input.java
./P2.sh input.java output.csv
```

Regular Expressions

A brief description of the regular expressions (in order of appearance on the official website) used in the `Flex` file has been provided below, wherever felt necessary-

- The RegEx for a "Whitespace" is straightforward; its definition does not include "Line Terminators" as they have been defined separately.
- In the implementation of RegEx for "single line comments", the line terminators are included for ease. It does not compromise correctness, since both of them are to be simply eaten up.
  For "multi-line comments", a single RegEx has been used for implementation which was obtained by repeatedly compressing the rules available on the official website, until a single rule was left.
- The regular expressions used for "Digits" and "HexDigits" was made after understanding the use-cases of the underscore character "_" in numeric literals, which is that a number should neither start, nor end with an underscore.
- In "Character" literals, escape sequences have not been handled.

- For "text blocks", `Start conditions` have been used, provided by Flex.
  After seeing three consecutive double quotes, the state `TEXT_BLOCK_WS` is entered which keeps looking for "Whitespace", and transitions to `TEXT_BLOCK_TB` on seeing a "Line Terminator". If any other character is seen, an error is reported, since multiline strings are not allowed. In the state `TEXT_BLOCK_TB`, characters are read until three successive double quotes are found again, in which case, the text block is complete.
  To keep track of the content of the "text block", a global string variable, "`text_block`" is used, and for error reporting purposes, a global integer variable, "`tb_start`" is used to remember where the "text block" had started.

- Escape sequences have *not* been handled, since the assignment PDF said they could be avoided.

- The implementation of regular expressions for the rest of the tokens was straightforward.