CS1319: PLDI - Assignment 2

Hrsh Venket & Santripta Sharma

September 2023

Explanation of Lexer

Below is the code we have written for the lexer rules. This explanation therefore is our attempt to explain our choices by explaining how our code and the given rules for the lexer are equivalent.

First let us consider the obvious cases, where we have written the regex for the lexer exactly as we have been given in the assignment.

```
KEYWORD char|else|for|if|int|return|void
PUNCT \[|\]|\(|\)|\{|\}|->|&|\*|\+|-|\/|%|!|\?|\<|... (and so on)
```

Here, we have simply written the regex as given in the assignment using a series of OR statements

The explanation for string literals also directly follows from the definition given in the assignment.

```
ESCAPE \\’|\\\?|\\\"|\\\\|\\a|\\b|\\f|\\n|\\r|\\t|\\v
STRCHAR [^\"\\\n]|{ESCAPE}
STRLIT \"{STRCHAR}*\"
```

We define string characters as either the escape sequence or a member or the character set except for the double quote, the backslash and the newline character. We do this as we have done for the rest of the lexer, using ^ to denote 'everything except'.

String literals are defined as 1 or more string characters within double quotes.

Two less obvious cases are the single and multiline comments. For the single line comments, we have written the regex as follows:

```
COMMENTSINGLE \/\/([^\n])*\n
```

Here, `\/\/` denotes the opening `//` for a single line comment. Inside the comment, we allow any characters except for the newline operator. The `*` means we can have 0 or more of the any type of character. It must thereby terminate with a newline character `\n`.
In the case of multline comments, our code is as below:

```
COMMENTMULTI \/\*([^\*]|\*[^\/])*\*\/
```

The multilinecomment starts with /*, followed by any number of characters, but where the character can be a * only if the next character is anything else besides a /. Finally, the comment is closed by */. This ensures that comments end on the first close multiline (*/) encountered.

Now we can consider the less direct cases.

```
IDENT [a-zA-Z\_][0-9a-zA-Z\_]*
```

By definition, the identifier cannot start with a digit. Therefore, we define that the identifier must start with an identifier non-digit, `[a-zA-Z\_]`. This is followed by 0 or more identifier characters, `[0-9a-zA-Z\_]`, which can be digits or non-digits

```
CHAR [^\\'\n]|{ESCAPE}
CONST ([\+-]?[1-9][0-9]*)|[0-9]+|'{CHAR}+'
```

Here, we describe const, using some of the definitions given in the assignment. We define `CHAR` as any character except for the single quote, backslash, or the newline character OR an escape sequence. We define `CONST` as the constant, which can be a number, a character, or a multicharacter literal (1 chars).

Besides character constants, we also define the integer constant as an optional sign, followed by a non-zero digit optionally followed by 0 or more digits from 0-9 OR a sequence of one or more digits 0-9 (no sign).