**Peanut Lexer - CS323 Documentation**

1. **Problem Statement**

The first thing a compiler needs to do to source code is process it using a lexer. The lexer must read in the source code and identify what type of token is associated with each lexeme. The rules for this are defined in the Rat23F documentation.

1. **Usage**

***Method 1 - Drag-and-drop:***

1. Using the File Explorer, navigate to the directory that *lexer.exe* is located in.

2. Drag-and-drop your Rat23F source file (e.g. *test.rat*) onto the executable.

A computer screen shot of a computer

Description automatically generated

3. Open the generated output file (e.g. *test.rat.lexr*) in a text editor to see the results.

A screenshot of a computer

Description automatically generated

***Method 2 – Command line:***

1. Using the Command Prompt, navigate to the directory that *lexer.exe* is located in.

2. Run *lexer.exe* while passing the path to the Rat23F source file to process as the first argument.

3. See the output in the console or open the generated output file to see the results.

A screenshot of a computer

Description automatically generated

1. **Program design**

The source code can be found at <https://github.com/jiink/peanut-compiler/blob/main/src/lexer/lexer.go>

*Identifier*, *Real*, and *Integer* token types are each represented by their own respective finite state machines. The lexer handles control flow among these FSMs and detects the other, enumerable token types using basic logic. The symbol set for the FSMs is composed of possible categorizations of source code characters: *Letter* (*l*), *Digit* (*d*)*,* *Period* (*p*)*,* and *Special* (*s*).

The regular expressions (REs) for these token types are as follows:

Identifier: *(l(l|d)\*l)|l*

Real: *d+pd+*

Integer: *d+*

The nondeterministic finite state machines (NFSMs) for these token types are as follows:

Identifier:

A diagram of a complex curve

Description automatically generated

Real:

A diagram of a mathematical equation

Description automatically generated

Integer:

A diagram of a circle with arrows and a circle

Description automatically generated

The NFSM diagram for *Identifiers* happens to look like a flying saucer. In implementation as DFSMs, when a finite state machine receives one of these symbols, its next state is looked up from its transition table, stored as a 2D array of integers. Each state is represented by an integer that corresponds to the index of a row in the table. The 0th row is reserved as an “unrecoverable” state that is reached when an expected symbol is received at an unexpected time. When this occurs, or an unexpected symbol is received that results in undefined behavior, the FSM halts, and reports whether it is in an accepting state or not. If it halted in an accepting state, the string it iterated through is labeled according to its token type.

1. **Limitations**

Identifiers, reals, and integers are limited to 500 characters.

This limit is arbitrary and is placed simply to prevent any possibility of infinite loops in the finite state machines.

1. **Shortcomings**

None known.