JenPile
Part 1 of Compiler Project
CS323 Documentation

Problem Statement:

To write a lexical analyzer, using a FSM for the entire lexer, or using FSM for identifier, integer and real numbers.

The function lexer, should return a token when it is needed. The lexer should return a record, one field for the token and another field for the actual value (lexeme) of the token. The main program should read in a file containing the source code given to generate tokens and write the results to an output file.

How to Use JenPile:

With a file input:
jenni.exe -c filename
Example: jenni.exe -c HelloWorld.jen

Without a file input, to type on the console line:
jenni.exe

Sample Test Files Included:
HelloWorld.jen
SampleInputFile.jen
ShortTestCase.jen

Design Of Program:

Program
Uses argos to pass in the file name, if the file to compile is null, then input is collected from the console. Prints the finished token list.

InputCollector
Reads in a file, or from the Console, line by line. Places each line read in a List structure until an empty line is reached.

TokenType
Contains the token types as an enum. The Token Types are none, keyword, identifier, separator, operator, integer, float and undefined.

Token
Contains the struct for the token type, value pair

TokenDictionary
Contains a Dictionary of given keywords and operators paired with their token types. The token types are the ones identified in the given keyword and operator list.

Lexer
Using the List from the InputCollector class, for each char in each line; first it checks for comments. If a ! is found, characters are not appended until another ! is reached.

Then the Lexer iterates through each character, appending each character to evalLine until it matches a separator. When a separator is found, the evalLine is checked to see if it is in the TokenDictionary. If it is not, evalLine is checked against each Regex pattern until a match is found. Regex is a regular expression Finite State Machine that does pattern matching for the defined patterns. The defined pattern types are for separators, identifiers, floats and integers. If a token is not identified, evalLine assigns it the undefined token.

Lastly, the token type and its value are paired together and added to a new List, along with the separator and it's token type. A list was chosen so it would be easier to iterate backwards and forwards during the Syntax stage. The List currently prints to the console with the Token Lexeme pair.

Limitations:
When read in from the console, if two identifiers are typed on two lines with no separator, it will read them together as one lexeme.

The = sign can be both an operator and a seperator, so when used with no spaces, it is classified as an undefined token. When space is given on both sides of the =, it is correctly identified as an operator.

Shortcomings:
<None>

FSM diagram