CSci 365: Organizations of Programming Languages

Name: ________________________

Assignment 2: Lexical Analyzer (100 + 10 (optional))

– Read the submission instructions.

Q1. [10] Design/draw a state transition diagram to recognize one form of the comments of the C-based programming languages, those that begin with /* and end with */. At the end, the transition will return 'COMMENT' if it's in the correct form; otherwise, it'll return an error message of 'SLASH CODE'. For the labels of the states, use q0, q1, q2, …, etc.

Assume that a comment is given in the same line and ends with a hidden 'EOL' symbol to indicate the end of the line. e.g.) /* a comment */EOL

Note: Do **not** include the utility functions 'getChar' and 'addChar' in the transition.

Q2. [10, optional] Design/draw a state diagram to recognize a float number, e.g.) **.**… *. It returns 'float'.
e.g.) float number: 2.0, 20.0101, 0.0, etc.

Q3. [20] Programming

Write a Python program to test the code to implement the state diagram of Q1.
Test your program with the following inputs and give the outputs.

Correct Input:
    Input 1: /* this is a comment */
Incorrect inputs:
    Input 2: // this is a comment //
    Input 3: // this is a comment */
    Input 4: /* this is a comment /*
    Input 5: */ this is a comment */
    Input 6: */ this is a comment */
    Input 7: /* this is a */ comment */

Hin

Note: Refer to the codes in 'front.c' that identify the identifiers and integer literals.

Q4. [70] Programming

Implement the lexical analyzer to handle the short program below. Write it in Python or in Java – **not in C** because its codes are available in the textbook.

Input **file**: a program below.

```
input(a)
input(b)
input(c)
total = a + b + c  /* get a sum of three inputs */
average = total / 3  /* compute an average */
print(total)
print(average)
```

<u>Output display on the **screen**</u>:

A list of (type of a token, ‘,’, lexeme)s. Each list of (token, lexeme) will be printed in the new line.

e.g.) (<ADD_op>, +)
(<ID>, sum)
etc.

e.g.) For the above input program:

```
<input>, input
<lparen>, (
<id>,    a
 …
```

<u>Lookup Table of the Token, special characters and words</u>:

Token	**Value/lexeme**	**Token**	**Value/lexeme**
<input>	input	<output>	print
<id>	identifier	<number>	integer, float
<lparen>	(	<rparen>	)
<add_op>	+, -	<mult_op>	*, /, //, %
<rel_op>	<, >, <=, >=, ==, !=	<assign_op>	=
<comment>	comment	<error>	error
reserved words	input, output, if, else, begin, end, while, for		
<input>	**input**	<output>	**print**
<if>	**if**	<else>	**else**
<begin>	**begin**	<end>	**end**
<while>	**while**	<for>	**for**

<u>Separator</u>:

whitespace, line_feed.

<u><id> token</u>: use the state diagram in the textbook

– starting with upper/lower letter, followed by any of upper/lower letter/digit

<u><number> token</u>:

If integer, starting with a non-zero digit, followed by digit(s)

If float, starting with a non-zero digit, followed by digit(s), a decimal point(.), followed by digit(s).

- *at least one* digit before and after a decimal point

<error> token:

If it detects any non-valid token which is not in the lookup table, write an <error> token and the invalid message with the value.

e.g.) && in the program → <error>, 'Invalid token' for &&