Habib University

CS 421 (Compiler Design and Construction) – FALL 2022

Assignment# 01 (Graded) – Total points: 10. Weightage in the final grading: 10%.

Announced: September 2, 2022. Due Date: Sunday, September 11, 2022, 11:55 pm (PKT) on Canvas.

Description: This assignment has two parts. Part I has three points. Part II has seven points and is implementation based. You can submit it in **pairs** or **individually.**

Part I: Submit your solution in a single PDF.

1. **[1 point]** Consider the following *C* code snippet:

```
int w, x, y, z;
int i = 4; int j = 5;
{
      int j = 7;
      i = 6;
      w = i + j;
}
x = i + j;
{
      int i = 8;
      y = i + j;
}
z = i + j;
```

   Indicate the values assigned to **x** and **z** (boldface).

2. **[1 point]** What is printed as output when the following *C* code is executed?

```
#define a(x + 1)
int x = 2;
void b() {x = a; printf("%d\n", x);}
void c() {int x = 1; printf("%d\n", a);}
void main() {b(); c();}
```

3. **[1 point]** Construct a regular expression to recognize *currency* numbers in dollars. It should be a positive decimal number rounded to the nearest one-hundredth. Currency numbers begin with the dollar sign $, have commas separating each group of three digits to the left of the decimal point, and end with two digits to the right of the decimal point, for example, $8,937.43 and $7,777,777.7.

Part II: Submit the answers in a single PDF.

Implement a lexical analyzer using C/C++ or Python for a hypothetical language ***TUPLE*** (The Ultimate Programming LanguagE). You may submit in pairs (any one member may submit). Your lexical analyzer must be able to read text stream from the three test files (code written in TUPLE with extension *.tpl*) and output a stream of tokens **<class, value>** (as per the given lexical specification), output the symbol table, and a file containing any detected lexical errors.

Reference: Please refer to Section 2.6, 3.1-3.5, 3.8 and the appendix of the 'Dragon Book' (2nd Ed.) for conventions and guidelines for implementing a lexical analyzer. Loudon's book is also a good resource.

Requirement: Your submission must be in a ZIP file named *Member_1-Member_2.ZIP* and must contain the following files:

1. **Source file(s):** of a working and correct lexical analyzer implemented in C/C++ or Python; along with any other auxiliary files. Your code must contain any necessary comments. Your implementation must be able to read from the given test files containing *lexemes* and generate the following output files. Your implementation should be able to recognize *lexemes* and output all specified tokens even if they are absent from the test files.

2. **Output files:** Your lexical analyzer should output three separate files for each of the given test files:

   a. A file with the same name as the input file but with an **extension (*.out*)**. This file should contain the **token stream** generated on reading the input file, in the form used in the 'Dragon Book', e.g., <if><ID, *ptr_to_symbol-table*><REL_OP, EQ><NUM, *ptr_to_symbol-table*><then>…

   b. A file with the same name as the input file but with an **extension (*.err*)**. This file should contain any error(s) detected by the lexical analyzer while scanning the input file. You may refer to the Draon Book pg. 114 and other sources about ***lexical errors.*** Your implementation should report the error, e.g., unidentified symbol(s), error in float *31.c*; missing closing quotation when recognizing a string literal; closing symbols missing when recognizing comments and write them in the error file together with **line number** at which the error was found: e.g.,

```
<line#>          <error_found>
34             $ (unrecognized symbol)
```

   c. A file with the same name as the input file but with an **extension (*.sym*)** that outputs the symbol table in a tabular format, i.e., each symbol table entry per line. See Section 2.7 and Chapter 3 for reference and conventions used.

Grading: Here is a breakup of the 07 points (07% of the final grading) for Part II:

Requirement	Point(s)
Lexical analyzer (source file/s) producing correct token stream as output (see 2.a.).	03
An output file containing lexical errors detected (see 2.b.).	02
Output file showing symbol table entries (see 2.c.).	02
Total	**07**

The TUPLE (The Ultimate Programming LanguageE) Lexical Specification

Keywords: `and, break, continue, else, false, for, if, mod, not, or, then, true, void, while`

Identifiers: Must begin with a letter followed by any sequence of letters, digits, and underscore (_).
Token: $< id, pointer\ to\ symbol\ table\ entry >$.

Data Types: `bool, char, int, float.` Token: $< dt, data\ type\ found >$.

Punctuators/Symbols: = `{ } ( ) ; [ ] ' " , .`

Number constants: Identify **signed numbers** and evaluate their values. Token: $< num,\ value >$.

String literals : `"any sequence of UTF-16 characters".` Token: $< literal,\ vlaue >$.

Character constant: `"at most one UTF-16 character".` Token: $< character,\ value >$.

Comment: Single line comment allowed only. `/$...$/`

Relational operators: `< <= > >= == != (you may use these instead: LT LE GT GE EQ NE).` Token: $< rel\_op,\ value >$.

Arithmetic operators: `+ - * / ^.` Token: $< +>, < ->, <*>, </>, < ^ >$.

Whitespaces: `blank newline tab.`