COMPILER PROJECT I 2022

The goal of the first term-project is to implement a lexical analyzer (a.k.a., scanner) as we've learned. More specifically, you will implement **the lexical analyzer for a simplified C programming language** with the following lexical specifications;

<Lexical specifications>

✓ Variable type: int or INT for a signed integer, char or CHAR for a literal string

✓ Signed integer: A non-empty sequence of digits, starting from a non-zero digit except an integer 0 (e.g., 0, -1, 10, 999)

✓ Literal string: any combination of digits, English letters, and blanks, starting from a symbol " and terminating with a symbol " (e.g., "Hello world", "My student id is 12345678")

✓ An identifier of variables and functions: a non-empty sequence of English letters and digits, starting from a English letter (e.g., i, j, k, abc, ab123, func1)

✓ Keywords for special statements: if or IF for if statement, else or ELSE for else statement, while or WHILE for while statement, return or RETURN for return statement

✓ Arithmetic operators: +, -, *, and /

✓ Assignment operator: =

✓ Comparison operators: <, >, ==, !=, <=, and >=

✓ A terminating symbol of statements: ; (e.g., a = 1;)

✓ A pair of symbols for defining area/scope of variables and functions: { and }

✓ A pair of symbols for indicating a function: (and)

✓ A symbol for separating input arguments in functions: ,

✓ Whitespaces: a non-empty sequence of ₩t, ₩n, and blanks

✓ Comments are not allowed

Based on this specification, you will 1) define tokens for a simplified C language, 2) make regular expressions which describe the patterns of the tokens, 3) construct a NFA for the regular expressions, 4) translate the NFA into a DFA, especially in the form of a table, and 5) implement a program which does a lexical analysis (recognizing tokens).

[IMPORTANT NOTE]

1. You MUST build regular expressions, NFAs, and DFAs by writing or drawing by hand.

(Do not use any kind of computer program like Lex for this procedure)

2. You can use C, C++, JAVA, or Python to implement your lexical analyzer.

3. Your lexical analyzer MUST run on Linux or Unix-like OS without any error. (Do not use Windows-only APIs and please test your program on Linux machines before submission)

5. Your lexical analyzer should work as follows;

✓ **The execution command of your lexical analyzer:** lexical_analyzer <input_file_name>

✓ **Input:** A program written in a simplified C programming language

(You don't need to think about the syntax of the program yet)

✓ **Output:** <intput_file_name.out>

■ (If an input program has no error) A symbol table which stores the information of all tokens including their names and optional values

◆ This output will be used as an input of your next term-project (syntax analyzer)

■ (Otherwise) An error report which explains why and where the error occurred (e.g., line number)

Input: test.c		Output: test.out	
int func(int a) { return 0; }		INT	
		ID	func
		LPAREN	(

		…	…	

6. Do not include "WHITESPACE" tokens in the output token list

7. There will be some issues with the symbol -. Please consider the syntax of the given code to address this.

Term-project schedule and submission

- ✓ **Deadline: 4/29, 23:59 (through an e-class system)**
  - ■ For a delayed submission, you will lose 0.1 * your original project score per each delayed day
- ✓ Submission file: team_<your_team_number>.zip or .tar.gz
  - ■ The compressed file should contain
    - ◆ The source code of your lexical analyzer with detailed comments
    - ◆ The executable binary file of your lexical analyzer
    - ◆ Documentation (the most important thing!)
      - ● It must include 1) the definition of tokens and their regular expressions, 2) the DFA transition graph or table for recognizing the regular expressions, 3) all about how your lexical analyzer works for recognizing tokens (for example, overall procedures, implementation details like algorithms and data structures, working examples, and so on)
    - ◆ Test input files and outputs which you used in this project
      - ● The test input files are not given. You should make the test files, by yourself, which can examine all the token patterns.
- ✓ If there exist any error in the given lexical specification, please send an e-mail to hskimhello@cau.ac.kr