Alexandria University
Faculty of Engineering
Computer and Systems Engineering Dept

Programming languages and compiler
Assigned: Wednesday 14th Dec, 2022
Due: Wednesday 28th Dec, 2022

Assignment 2

Assignment will be into 3 phases.

Objective of Whole Assignment

- *It is required to develop a suitable Syntax Directed Translation Scheme to convert Java code toJava bytecode, performing necessary lexical, syntax and static semantic analysis (such astype checking and Expressions Evaluation).*
- *Generated bytecode must follow the standard bytecode instructions defined in Java Virtual Machine Specification*
  http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html
  http://en.wikipedia.org/wiki/Java_bytecode

Phase 2: Parser Generator

Objective

This phase of the assignment aims to practice techniques for building automatic parser generator tools.

Description

1- *Your task in this phase of the assignment is to design and implement an LL (1) parser generator tool.*

2- *The parser generator expects an LL (1) grammar as input. It should compute First and Follow sets and uses them to construct a predictive parsing table for the grammar.*

3- *The table is to be used to drive a predictive top-down parser. If the input grammar is not LL (1), an appropriate error message should be produced.*

4- *The generated parser is required to produce some representation of the leftmost derivation fora correct input.*

5- *If an error is encountered, a panic-mode error recovery routine is to be called to print an error message and to resume parsing.*

6- *The parser generator is required to be tested using the given context free grammar of a small subset of Java. Of course, you have to modify the grammar to allow predictive parsing.*

7- *Combine the lexical analyzer generated in phase1 and parser such that the lexical analyzer is to be called by the parser to find the next token. Use the simple program given in phase 1 to test the combined lexical analyzer and parser..*

Java CFG

```
CLASS_DECL::= MODIFIER class id { CLASS_BODY }
CLASS_BODY::= DECLARATION | ASSIGNMENT | METHOD_LIST | Epsilon
METHOD_LIST::= METHOD_DECL | METHOD_LIST METHOD_DECL
METHOD_DECL::= MODIFIER PRIMITIVE_TYPE id() { METHOD_BODY }
METHOD_BODY:: = STATEMENT_LIST
STATEMENT_LIST::= STATEMENT | STATEMENT_LIST STATEMENT
STATMENT ::= DECLARATION
                    | IF
                    | WHILE
                    | ASSIGNMENT
DECLARATION ::= PRIMITIVE_TYPE IDENTIFIER;
PRIMITIVE_TYPE ::= int | float
MODIFIER ::= public | private | protected
IF ::= if ( EXPRESSION ) { STATEMENT} else { STATEMENT }
WHILE ::= while ( EXPRESSION ) { STATEMENT }
ASSIGNMENT ::= IDENTIFIER = EXPRESSION;
EXPRESSION ::= NUMBER
                          | EXPRESSION INFIX_OPERATOR EXPRESSION
                          | IDENTIFIER
                          | ( EXPRESSION )

INFIX_OPERATOR ::= + | - | * | / | %| < | > | <= | >= | == | !=| | | &&
```

CFG Input File Format

1- CFG input file is a text file.
2- Production rules are lines in the form LHS ::= RHS
3- Production rule can be expanded over many lines.
4- Terminal symbols are enclosed in single quotes.
5- \L represents Lambda symbol.
6- The symbol | is used in RHS of production rules with the meaning discussed in class.
7- Any reserved symbol needed to be used within the language, is preceded by an escape backslashcharacter.

Input file example:

```
# METHOD_BODY = STATEMENT_LIST
# STATEMENT_LIST = STATEMENT | STATEMENT_LIST STATEMENT
# STATEMENT = DECLARATION
                        | IF
                        | WHILE
                        | ASSIGNMENT
# DECLARATION = PRIMITIVE_TYPE ‘id’ ‘;’
# PRIMITIVE_TYPE = ‘int’ | ‘float’
# IF = ‘if’ ‘(‘ EXPRESSION ‘)’ ‘{‘ STATEMENT ’}’ ‘else’ ‘{’ STATEMENT ‘}’
# WHILE = ‘while’ ‘(‘ EXPRESSION ‘)’ ‘{‘ STATEMENT ‘}’
# ASSIGNMENT = ‘id’ ‘=’ EXPRESSION ‘;’
# EXPRESSION = SIMPLE_EXPRESSION
                        | SIMPLE_EXPRESSION ‘relop’ SIMPLE_EXPRESSION
# SIMPLE_EXPRESSION = TERM | SIGN TERM | SIMPLE_EXPRESSION ‘addop’ TERM
# TERM = FACTOR | TERM ‘mulop’ FACTOR
# FACTOR = ‘id’ | ‘num’ | ‘(‘ EXPRESSION ‘)’
# SIGN = ‘+’ | ‘-‘
```

Parser Output File Format

Your program should output the predictive parsing table of the generated parser in a format of your choice as well as the leftmost derivation sententials one per line (like the following format).

Output file example for the given test program:

```
int x;
x = 5;
if (x > 2)
{
      x = 0;
}
```

```
METHOD_BODY
STATEMENT_LIST
STATEMENT_LIST STATEMENT
STATEMENT_LIST STATEMENT STATEMENT
STATEMENT STATEMENT STATEMENT
DECLARATION STATEMENT STATEMENT
PRIMITIVE_TYPE IDENTIFIER; STATEMENT STATEMENT
int IDENTIFIER; STATEMENT STATEMENT
....to be continued
```

Bonus Task

Automatically eliminating grammar left recursion and performing left factoring before generating theparser will be considered a bonus work

Notes

1. *Implement the project using C++.*
2. *Each group consists of 4 students.*
3. *Requirements:*
   1- *Your executables and source code.*
   2- *A project report: make sure that your report contains at least the following:*
      a. *A description of the used data structures.*
      b. *Explanation of all algorithms and techniques used*
      c. *The resultant transition table for the minimal DFA.*
      d. *The resultant stream of tokens for the example test program.*
      e. *Any assumptions made and their justification.*
4. *Submit your work including the code and the report in a zip file to this form: https://docs.google.com/forms/d/e/1FAIpQLSd7Ww1iBPHXJLS0L1ycPmFdwZc1vS36pH_QD9_-Dc8_yDH2hA/viewform?usp=sf_link*

Good luck