CS 415 Compilers: Problem Set 4

Spring 2022

Due date: Monday, March 28, 11:59pm

Problem 1 – Parse trees, derivations

Consider the following grammar with start symbol A:

```
1    A ::= ( B )   |
2          a
3    B ::= B , A   |
4          A
```

1. Give parse trees for the sentences **(a, a)** and **((a, a), a)**.
2. Construct a leftmost and a rightmost derivation for the sentence **((a, a), a)**.

Problem 2 - LL(1)

Assume the following grammar of a simple, prefix expression language.

```
Program ::= Stmtlist .
Stmtlist ::= Stmt NextStmt
NextStmt := ; Stmtlist | epsilon
Stmt ::= Assign | Print
Assign ::= ID = Expr
Print ::= ! ID
Expr ::= + Expr Expr |
         - Expr Expr |
         * Expr Expr |
         ICONST
ID ::= a | b | c
ICONST ::= 1 | 2 | 3 | 4 | 5
```

Programs are written in a single line (no new line characters), and there are no blanks. This means that all tokens are exactly one character long, and that you don't have to worry about line breaks.

Te homework problem consists of the following subproblems:

1. Compute the *FIRST* and *FOLLOW* sets for the grammar.

2. Compute the LL(1) parse table for the resulting grammar. Is the grammar LL(1) or not? Justify your answer.

3. If the resulting grammar is LL(1), show the behavior of the LL(1) skeleton table-driven parser as a sequence of states [stack content, remaining input, next action to be taken] on sentence `c=3;!c.` .

Problem 3 - Recursive Descent Parsing

If the grammar is LL(1), write two recursive descent parsers in C, one implementing an interprete, and one a compiler. Note that each "token" is exactly one character, and that there are no spaces within a program. You may assume that the program is syntactically correct, so no error detection or recovery is required. You should submit the source codes for both parsers, `interpreter.c` and `compiler.c`, Please follow the posted submission instructions.

1. Write an interpreter for the language of Problem 2. This means, that given an input, the parser "executes" the program and prints its results. For example, the program

   ```
   a=+*345;!a.
   ```

   will prints "17" on the standard output.

2. Write a compiler for the language of Problem 2. This means, that given an input, the parser will generate an ILOC program in file iloc.out. For the example program above, the ILOC code in file iloc.out should be:

   ```
   loadI 1024 => r0
   loadI 3 => r1
   loadI 4 => r2
   mult r1, r2 => r3
   loadI 5 => r4
   add r3, r4 => r5
   storeAI r5 => r0, 4
   outputAI r0, 4
   ```

assuming that variable “a” is stored at memory location base_address + offset 4. When running iloc.out on the ILOC simulator, the printed value should be “17”. Your program should assign each value a new register number (fresh virtual register), starting with register 1. Your generated program should run correctly on the ILOC simulator.