Computer Science 433
Programming Languages
The College of Saint Rose
Fall 2012

Program/Problem Set 3: Syntax

Due: 11:59 PM, Tuesday, September 25, 2012

In this assignment, you will be completing some problems related to syntax, grammars, and parse trees.

You may work alone or with a partner on these problems.

Textbook Problems

Question 1: Do Exercise 12 on p. 164 of Sebesta. (2 points)

Question 2: Do Exercise 13 on p. 164 of Sebesta. (4 points)

Question 3: Do Exercise 14 on p. 164 of Sebesta. (4 points)

Reverse Polish Notation

Reverse Polish notation (RPN) is a common name for postfix mathematical expressions. Such notation lends itself to a stack-based evaluation and is used by the PostScript printer language and some old calculators. It has the advantage that it can specify any expression that can be specified in the usual (infix) notation without the need for parentheses to enforce order of operations.

If we are given an expression in RPN we would evaluate it as follows.

- Start with an empty stack.
- Evaluate the expression from left to right. Each step of the way:
  - if a number is encountered, push it onto the stack
  - if an operator is encountered, pop 2 numbers from the stack, apply the operator to those numbers, push the result onto the stack
- If the expression is valid, there should be a single number left on the stack.

So for the expression

```
9 7 12 - *
```

we start with the empty stack. We then push `9`, push `7`, push `12`, so our stack consists of `[ 9 7 12 ]`. We encounter the `-` operator, pop the `12` and the `7` from the stack, calculate `7 - 12 = -5` and push the result, giving a stack of `[ 9 -5 ]`. We then encounter the `*` operator, pop the `-5` and `9` from the stack, calculate `5 * -9 = -45` and push the result, giving a stack of `[ -45 ]`. Now there are no more tokens on the input, so the value at the top of the stack contains our result, `-45`.

Question 4: Evaluate the following RPN expressions or state that it is an invalid RPN expression and why. Assume integer division rules apply where appropriate. (1/2 point each)

```
5 8 19 + *
2 3 + 5 7 * /
7 * 12 + 9 / 3
23
2 3 + 5 7 * / 3 4 + * 1 -
9 9 * 8 7 * * 5 5 * * 4 -
```

Question 5: Convert the following infix expressions into their RPN equivalents. (1/2 point each)

```
5 - 3 * 2 + 7
10 * 3 * 9 / 4
(5 + 8) / (9 -3)
5 + 8 / 9 - 3
```

Question 6: Use BNF to write a grammar for reverse Polish notation that includes the `+`, `-`, `*`, and `/` operators. (8 points)

Question 7: Using your grammar from the previous question, draw a parse tree for the RPN expression (4 points)

```
3 4 + 9 3 - 4 * *
```

A Grammar for BASIC

Considered this grammar for a subset of the BASIC language.

```
   <program> => <lines>
     <lines> => <line> | <line> <lines>
      <line> => <line-number> <stmt> \n
<line-number> => integer-literal
      <stmt> => REM string-literal
              | PRINT <print-expr>
              | INPUT <variable>
              | LET <assignment>
              | END
  <variable> => <integer-var> | <string-var>
```

```
<integer-var> => integer-variable-name
 <string-var> => integer-variable-name$
 <print-expr> => "string-literal" | <variable>
 <assignment> => <integer-var> = integer-literal
              | <string-var> = "string-literal"
```

Question 8: Construct a leftmost derivation and corresponding parse tree for the program:

```
10 REM THIS IS FUN!
20 LET X = 8
30 PRINT X
```

Three types of BASIC statements that are not included are the `IF/THEN` construct, the standard `GOTO` statement, and the `IF/GOTO` construct

An `IF/THEN` looks like this:

```
30 IF X>5 THEN LET X = X - 5
```

A `GOTO` statement looks like this:

```
70 GOTO 10
```

An `IF/GOTO` statement looks like this:

```
100 IF Y<>Z GOTO 150
```

Question 9: Augment the BNF grammar above to include the BASIC `IF/THEN` construct, the `GOTO` statement, and the `IF/GOTO` construct. For simplicity, assume that the only conditions permitted for the boolean condition on the `IF/THEN` and `IF/GOTO` are numeric comparisons of integer variables and integer literals and that only the standard comparison operators are supported (`=` for equality, `<`, `>`, `<=`, `>=`, and `<>` for inequality). (8 points)

Submission

To submit this assignment, send a single PDF file with your responses as an attachment to *teres-coj@strose.edu* by 11:59 PM, Tuesday, September 25, 2012.

Please include a meaningful subject line (something like “CS433 Program/Problem Set 3 Submission”).

Grading

This assignment will be graded out of 35 points, as broken down by the numbered questions throughout this document.