CS400 Compiler Construction
Fall 2008, Dr. Sheldon Liang

Homework & Quizzes #02

(25 points)

Due Date: One Week Away from today
(*Look at schedule in the syllabus*)

Your Name: _____ Your Score: _____

Objectives (*Context-free grammar*):

☞ A Context Free Grammar (CFG) is a set of recursive rewriting rules (or productions) used to generate patterns of strings. To be familiar with parse tree of a CFG helps to understand the principle of how to verify if a variety of expressions or statements if grammatically right or wrong:

- the root s labeled by the start symbol
- Each leaf is labeled by a terminal
- Each interior node is labeled by a nonterminal.

Questions and points

Number of questions:	[10]
Positive points per question:	[2.5]
Negative points per question:	[0.5]

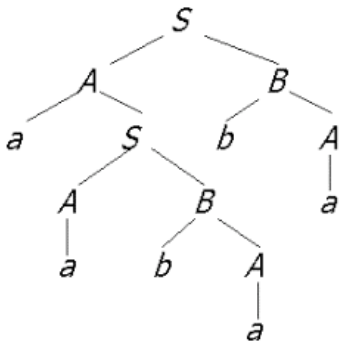
Make sure your name is on this handout before turning it in

⇨ Since we stress "learning through lecturing and reading", some questions designed for homework & quizzes stimulate students to listen, think, and read the text in our lectures. Be careful to find answer from the handouts and text or lab-testing (that is, seeking answer through program). ↵

Chapter2, 4: Analysis and Parsing

Grammatical Derivations and Parse Trees
Leftmost, rightmost derivations and their relationship to parse trees of a CFG.
Postfix form, syntax trees, translation of simple expressions.

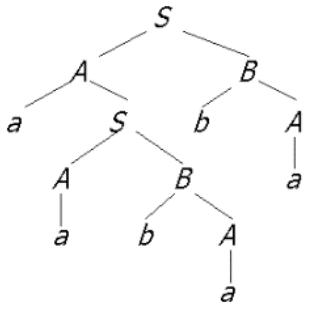
1. The parse tree below represents a rightmost derivation according to the grammar S → AB, A → aSla, B → bA.



Which of the following is a right-sentential form in this derivation?

☐ a) aaBba ☐ b) aSB ☐ c) Aba ☐ d) aababA

2. Here is a parse tree that uses some unknown grammar G.



Which of the following productions is surely one of those for grammar G?

☐ a) B → b
☐ b) S → B
☐ c) S → AbA
☐ d) B → bA

3. Here is a grammar for postfix expressions using the common four binary arithmetic operators:

S → SS+ | SS- | SS* | SS/ | a

Find a leftmost derivation for the terminal string aa-aa*/. Then, identify one of the steps (left-sentential forms) in the derivation from the list below.

☐ a) aS-S/ ☐ b) aa-Sa*/ ☐ c) aa-SS/ ☐ d) Sa/

4. Here is a grammar for postfix expressions using the common four binary arithmetic operators:

S → SS+ | SS- | SS* | SS/ | a

Find a rightmost derivation for the terminal string aa/a+a-. Then, identify one of the steps (right-sentential forms) in the derivation from the list below.

☐ a) aa/S+S- ☐ b) aS+a- ☐ c) Sa+a- ☐ d) aS-

5. The parse tree below represents a leftmost derivation according to the grammar S → AB, A → aS|a, B → bA.

Which of the following is a left-sentential form in this derivation?

☐ a) aaBB ☐ b) aaSBB ☐ c) aAbaba ☐ d) aSbA

6. Translate the infix expression

((a+b)-(c*d))+e

to prefix form. Then, identify the pair of symbols that appear consecutively in your prefix expression.

☐ a) *- ☐ b) b+ ☐ c) de ☐ d) +e

7. Here is a postfix arithmetic expression:

ab+c-d+*e+

The operator - appearing between c and d can represent either unary or binary minus. Normally, the other operators, + and *, are only binary operators. Consider translating the postfix expression into infix, both with - as unary and - as binary. Then, identify the true statement below.

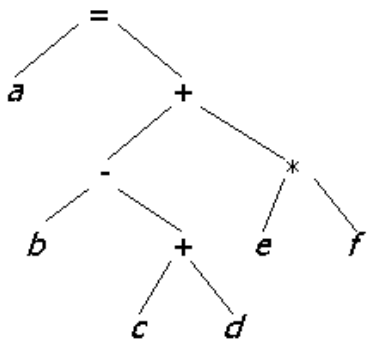
☐ a) - can be unary, and (a+b) is a subexpression of the infix expression.

☐ b) - in this postfix expression can be either unary or binary.

☐ c) - can be binary, and ((a+b)-c) is a subexpression of the infix expression.

☐ d) - in this postfix expression can be binary but cannot be unary.

8. The following syntax tree:



Represents the assignment

```
a = (b - (c+d)) + e*f;
```

Convert this tree to (op, arg1, arg2, result) quadruples, following these rules:

1. Evaluate the right subtree of a node before the left subtree.
2. Use temporaries t1, t2,..., in that order.

Then, identify from the list below, the one quadruple that would appear in your translation.

- ☐ a) (-, t2, b, t3)
- ☐ b) (+, t2, c, d)
- ☐ c) (+, t2, t1, t3)
- ☐ d) (+, c, d, t2)

9. Suppose we use the following simplified translation scheme for assignment statements, where no error checking is performed.

```
E → id = E      { gen(id.name() "=" E.place) }
E → E1 + E2     { E.place = newTemp();
                  gen(E.place "=" E1.place "+" E1.place) }
E → E1 * E2     {E.place = newTemp();
                 gen(E.place "=" E1.place "*" E1.place) }
E → - E1        { E.place = newTemp();
                  gen(E.place "=" "-" E1.place) }
E → ( E1 )      { E.place = E1.place }
E → id          { E.place = id.name() }
```

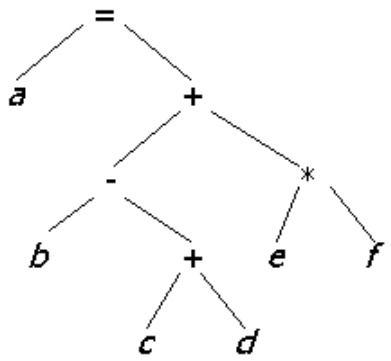
Here, name() is a method that returns the name of an id token from the symbol table, and newTemp() is a function that returns a new temporary. Assume that newTemp() returns T1, T2,... in order, when in it is called. Function gen(...) emits a line with a three-address statement, using the constituents inside parentheses, in order.
Apply this translation scheme to the assignment:

```
x = a * (-b + c)
```

To be specific, assume that the parse is bottom-up, and a rule is normal precedence for operators, to resolve conflicts. Then, identify the three-address statement that appears in your result.

- ☐ a) T2 = a * T1
- ☐ b) T3 = T1 + c
- ☐ c) T3 = a * T2
- ☐ d) T3 = T2 + c

10. The following syntax tree:



Represents the assignment

```
a = (b - (c+d)) + e*f;
```

Convert this tree to (op, arg1, arg2) triples, following these rules:

1. Evaluate the right subtree of a node before the left subtree.
2. Number the instructions (1), (2),...

Then, identify from the list below, the one triple, with its instruction number, that would appear in your translation.

- ☐ a) (1) [+, c, d]
- ☐ b) (3) [-, b, (2)]
- ☐ c) (4) [+, (2), (3)]
- ☐ d) (3) [-, b, d]