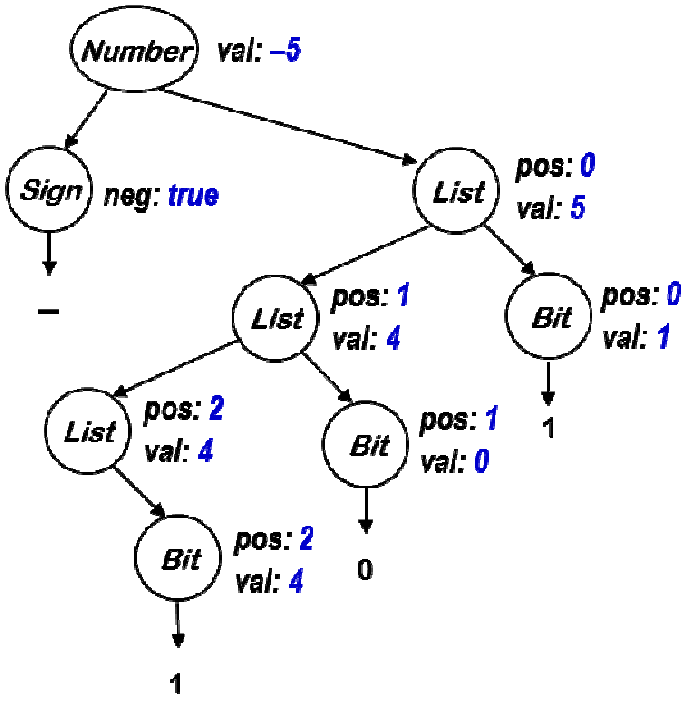
CMSC430 Spring 2014 Quiz 2 Solution

1. (6 pts) Attribute grammars

Consider the following attribute grammar for calculating values of signed binary numbers. Show how the attribute grammar can calculate the value for "-101" by annotating its parse tree, showing how each attribute is calculated.

Productions			Attribution Rules
Number	→	*Sign List*	*List.pos* ← *0* *If Sign.neg* *then Number.val* ← – *List.val* *else Number.val* ← *List.val*
Sign	→	+	*Sign.neg* ← *false*
	|	=	*Sign.neg* ← *true*
$List_0$	→	$List_1$ *Bit*	$List_1.pos$ ← $List_0.pos$ + 1 *Bit.pos* ← $List_0.pos$ $List_0.val$ ← $List_1.val$ + *Bit.val*
	|	*Bit*	*Bit.pos* ← *List.pos* *List.val* ← *Bit.val*
Bit	→	0	*Bit.val* ← *0*
	|	1	*Bit.val* ← $2^{Bit.pos}$



2. (8 pts) Syntax directed translation & type checking

   Consider the following grammar fragment for an OCaml-style if-then-else expression:

   exp → *IF exp THEN exp ELSE exp*

   Assume that the nonterminal *exp* has an attribute *type* that can be assigned a range of values, including *typeBoolean*, *typeInt*, *typeError*, etc. Add syntax-directed type checking rules to the grammar fragment to enforce the following:

   a. (2 pts) The guard of the IF expression must be *typeBoolean*. Otherwise call parser.msg("Boolean required").

   b. (2 pts) The THEN and ELSE expressions must have the same type. Otherwise call parser.msg("Else type mismatch").

   c. (4 pts) The type of the IF expression is assigned the type of the THEN expression if both (a) and (b) are true, else it is assigned *typeError*.

Answer:

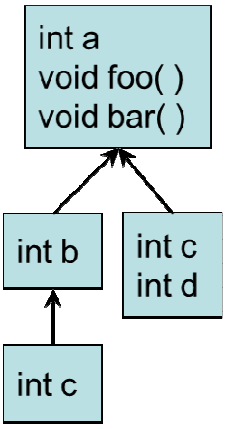
***exp*$_1$ → *IF exp*$_2$ *THEN exp*$_3$ *ELSE exp*$_4$**

```
{:
        exp1.type = exp3.type;
        if (exp2.type != typeBoolean) {
                parser.msg("Boolean required");
                exp1.type = typeError;
        }
        if (exp3.type != exp4.type) {
                parser.msg("Else type mismatch");
                exp1.type = typeError;
        }
:}
```

3. (6 pts) Symbol tables

Show the state of lexically nested symbol tables when the compiler reaches the point marked HERE in the following code fragment:

```
int a;
void foo( ) { int b; { int c;} }
void bar( ) { int c, d; /* HERE */ }
```



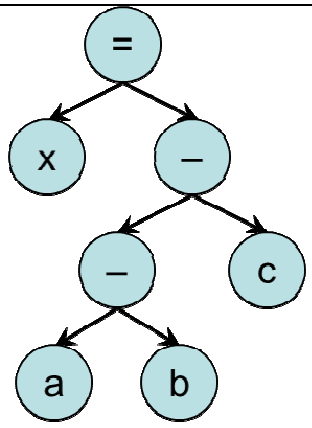
4. (10 pts) Intermediate Representations

Translate the following arithmetic expression into:

$$x = (a - b) - c$$

3-addr Instruction	Effect
load R1 x	R1 ← x
store x R1	x ← R1
add R1 R2 R3	R1 ← R2 + R3
sub R1 R2 R3	R1 ← R2 - R3
mult R1 R2 R3	R1 ← R2 * R3

Java Stack Code	Effect
nop	none
ldc_int c	push constant *c* onto stack
iload index(x)	push local variable X onto stack
istore index(x)	pop stack, store in local variable X
iadd	pop 2 elems off stack, add, push
isub	pop 2 elems, subtract top from 2nd, push
imult	pop 2 elems off stack, multiply, push

(2 pts) an abstract syntax tree (AST)	(4 pts) 3-address code	(4 pts) Java stack code
	**load R1 a** **load R2 b** **sub R3 R1 R2** **load R4 c** **sub R5 R3 R4** **store x R5**	**iload index(a)** **iload index(b)** **isub** **iload index(c)** **isub** **istore index(x)**