TEST TWO **NAME**:
Grammars, SPL ASTs and TPL **ID:**

QUESTION 1 (12 points)

Consider the following grammar:

(1) B → B "and" B	(2) B → B "or" B	(3) B → "!" F
(4) B → F	(5) F → <id>	(6) F → "true"
(7) F → "false"	(8) F → "(" B ")"	

where <id> is the identifier token

a. Encircle/mark the strings defined by the non-terminal B above: **[6 points]**

true	true false	x and true or y
y ! x	and x true	! (p and q)

b. Write a leftmost derivation that generates the string: "**x and y.**"
Label each arrow with the appropriate rule number. **[3 points]**

c. Write the rightmost derivation that generates the string: "**true or (x and y)**"
Label each arrow with the appropriate rule number. **[3 points]**

QUESTION 2 (TPL) (3 points)

Assuming variable x is stored in memory location 4, write the TPL code that computes:

```
x := x + 2;
```

QUESTION 3 (TPL) (3 points)

Consider the following sequence of TPL instructions:

```
STORE 0, R3
LABEL L1
GT R1, 0, R2
JMP0 R2, L2
ADDI R3, R1, R3
SUBI R1, 1, R1
JMP L1
LABEL L2
```

If the initial contents of R1 is integer **n > 0**, what's the final value stored in R3? Choose one from the list below:

n 1+2+…+(n-1)+n n*n 2n n+1 0

QUESTION 4 (10 points)

Suppose you want to add the RepeatUntil statement to the SPL defined by the following syntax:

RepeatUntilStm → "**`repeat`**" *Block* "**`until`**" "**`(`**" *Exp* "**`)`**"

Block → "**`{`**" *Statement*+ "**`}`**"

a. Define an appropriate Abstract Syntax for RepeatUntilStm**. [2 points]**

b. Write the JavaCC code that parses and creates the AST for *RepeatUntilStm*. You may assume that `block()` returns a list of statements `List<Stm>`**. [4 points]**

c. Translate the RepeatUntil statement into TPL, that is, provide a definition for genCode(RepeatUntilStm, stable) **[4 points]**