This document contains the solution for Test 2. I have added additional comments, useally preceded by the keyword **comment**.

Test 2 - Solution

1. Given the grammar presented in Question 1.

   a. The strings that can e derived by the non-terminal B of the grammar presented in question 1 are:

   - 'true' (YES)

     B $\xrightarrow{4}$ F $\xrightarrow{6}$ true

   - 'true false' (NO)
     comment: The only way to get true false is from F F, which can only be generated by B B, which cannot be derived from B.

   - 'x and true or y' (YES). A leftmost derivation:

     B $\xrightarrow{1}$ B and B $\xrightarrow{4}$ F and B $\xrightarrow{5}$ x and B $\xrightarrow{2}$ x and B or B $\xrightarrow{4}$ x and F or B $\xrightarrow{6}$ x and true or B $\xrightarrow{4}$ x and true or F $\xrightarrow{5}$ x and true or y

   - 'y ! x' (NO)

   - 'and x true' (NO)
     comment: 'and' can only be between two B's.

   - '!(p and q)' (YES). A rightmost derivation:

     B $\xrightarrow{3}$!F $\xrightarrow{8}$!(B) $\xrightarrow{1}$!(B and B) $\xrightarrow{4}$!(B and F) $\xrightarrow{5}$!(B and q) $\xrightarrow{4}$!(F and q) $\xrightarrow{5}$!(p and q)

   comment: The question didn't ask to explain your answer. I have added a complete derivation is the answer was YES, and additional comments to some of the NOs.

   b. Write the leftmost derivation that generates the string 'x and y'.
   B $\xrightarrow{1}$ B and B $\xrightarrow{4}$ F and B $\xrightarrow{5}$ x and B $\xrightarrow{4}$ x and F $\xrightarrow{5}$ x and y

   c. Write the rightmost derivation that generates 'true or (x and y)'.
   B $\xrightarrow{2}$ B or B $\xrightarrow{4}$ B or F $\xrightarrow{8}$ B or (B) $\xrightarrow{1}$ B or (B and B) $\xrightarrow{4}$ B or (B and F) $\xrightarrow{5}$ B or (B or y) $\xrightarrow{4}$ B or (F or y) $\xrightarrow{5}$ B or (x and y) $\xrightarrow{4}$ F or (x and y) $\xrightarrow{6}$ true or (x and y)

   comment: Some students could not differentiate between leftmost and rightmost derivations - hopefully the solution will clarify this. Other errors: some answers did not use the rule B $\xrightarrow{4}$ F. Finally, I was not very strict with the labelling of derivations - I will pay more attention when marking the exam.

   commentTry building the parse trees for the derivations above. Is the grammar ambiguous? If this is the case, which strings can be used to show this?

2. Assuming variable x is stored in memory location 4. Write the TPL code that computes `x := x + 2`.

   Answer: `ADDI $4,2,$4`

comment: The value of memory locations can be accessed by TPL instructions directly: you just need to place $ next to the address. ADDI can do the addition and perform a store. The code above is the shortest answer.

If we restrict operands to constants and registers then we would have to write:

```
STORE $4, R1
ADDI R1,2,R1
STORE R1, $4
```

3. If the initial contents of R1 is integer $n > 0$, what's the final value stored in R3?

```
STORE 0, R3      //  R3 = 0
LABEL L1         //  loop entry
GT R1, 0, R2     //  R2 =  R1 > 0
JMP0 R2, L2      //  if R2 is false (R1 <= 0) jump to L2 (exit)
ADDI R3, R1, R3  //  R3 = R3 + R1 (the initial value of R1 is n)
SUBI R1, 1, R1   //  R1 =  R1 - 1
JMP L1           //  go back to loop entry
LABEL L2        // loop exit
```

Answer: n + (n-1) + (n-2) + ... + 2 + 1

comment: The code above is equivalent to (pseudo-code):

```
R3 = 0;
R1 = n;
while (R1 > 0) {
  R3 = R3 + R1;
  R1 = R1 + 1;
}
```

4. The syntax of RepeatUntil:

   RepeatUntilStm $\longrightarrow$ `repeat` *Body* `until` (*Exp*)
   Body $\longrightarrow$ { *Statement+* }

   a. We only need to store the body of the statement (non-empty sequence of statements) and the condition (expression). Therefore, the abstract syntact tree representation can be wirtten as follows:

      RepeatUntilStm(List<Stm> body, Exp e)

   b. Write the JavaCC code that parses and creates the AST for the RepeatUntil statement.

```
Stm RepeatUntil() :
{ Exp e; List<Stm>  body;
}
{
  "repeat"   body = block()  "until"   "(" e=Exp() ")"
  { return new RepeatUntilStm(body,e); }
}
```

      assuming that there is a constructor for the class RepeatUntilStm that takes as arguments a list of statements and an expression, and that the non-terminal block() returns a new list of statements.

   c. Translate RepeatUntilStm into TPL.

      Given abstract syntax RepeatUntilStm(List<Statement> body, Exp e), a RepeatUntil-statement is executed as follows:

1) Execute each statement of body.
2) Execute test e. If it returns false, goto 1). If it returns true, end execution of statement.

Given the sequence above, the translation into TPL can be defined as follows:

```
Let:
    t = stable.getTemp(e)
    L1 = genLabel()
    s1,s2,...,sn = body

genCode(RepeatUntilStm(body,e)) =
    LABEL L1
    genCode(s1,stable)
    . . .
    genCode(sn,stable)
    genCode(e, stable)        - result of e is stored in t
    JMP0 t, L1                - if t is false, jump to L1
```