1. Problem 1 (25 Points)

Consider the following grammar:

A → B (',', B)+ => "a"
   → B

B → B '&' C => "b"
   → C

C → D '#' C => "c"
   → D

D → <identifier>

Write the skeleton of a recursive descent parser for this grammar, including 'BuildTree' statements that will build the AST bottom-up, for the original grammar.

Hint: See Powerpoint lecture 6.

Solution:
Proc A; { A → B (',', B)+ => "a"
         → B }
Var N: Integer;
B;
N := 1;
While Next_Token = T_, do
   Read (T_);
   B;
   N := N + 1;
Od
Build_Tree ("a", N);
end;

Proc B; { B → B '&' C => "b"
         → C }
C;
while Next_Token = T_& do
   Read (T_&);
   C;
Problem 2 (25 Points)

The following RPAL program is overly parenthesized; remove all superfluous parentheses, i.e. minimize the parentheses without changing the meaning of the program.

Also, format the program by writing it on several lines with the appropriate indentation.

\[
\text{let } (x = (a \cdot b) - c) \text{ in } ((e \land f) \rightarrow (g \rightarrow x \cdot (x^2) | y) \\
| (f(x) \text{ where } ((f(y) = y \cdot 2) \text{ and } (x = 2 \cdot (x^2))))))
\]

Solution:

\[
\text{let } x = a \cdot b - c \\
\text{in } \\
\qquad e \land f \rightarrow g \rightarrow x \cdot x^2 \\
\qquad \mid y \\
\qquad \mid (f(x) \\
\qquad \quad \text{where } (f(y) = y \cdot 2 \\
\qquad \quad \text{and } x = 2 + x \cdot 2)
\]
3. Problem 3 (25 Points)

Write, test, and debug an RPAL program that computes the "tuple reverse" function:

Rev(4,'hello',(3,4),true) = (true,(3,4),'hello',4).

Please write them on several lines with the appropriate indentation.

Solution:

The result may vary. Here is one example:

let Rev T = Prev (T,1)
    where rec Prev (T,N) = N gr (Order T) -> nil
         | (Prev (T,N+1) aug (T N))

in
Print (4,'hello',(3,4),true)

4. Problem 4

Explain the behavior of the following RPAL program, and the type of each identifier which appeared:

Let Prod N = P 1 N where rec P Cum N = N eq 1 -> Cum | P (Cum*N)
in Print (Prod 5 4 3 2 1)

Make an honest attempt at figuring it out BEFORE asking the RPAL interpreter.

Solution:

Function Prod takes an arbitrary number of arguments (one at a time) and multiplies them. The multiplication stops when the current argument is 1.

Types:
Prod / P / Print -Function
N(in Prod) / N(in P) / Cum -Integer