1. **Parser Construction:** (25 points)

Below is the portion of the RPAL grammar that handles tuple and conditional expressions. Write the pseudo-code for the corresponding portion of a recursive descent parser. Assume the existence of procedure B. Include in your parser the necessary “BuildTree” statements to build the Abstract Syntax Tree, in bottom-up fashion.

\[
\begin{align*}
T & \rightarrow T \ ( \ , \ T \ ) + \quad \Rightarrow \ ‘tau’ \\
& \quad \rightarrow T a \\
Ta & \rightarrow T a \ ‘aug’ \ T c \quad \Rightarrow \ ‘aug’ \\
& \quad \rightarrow T c \\
Tc & \rightarrow B \ ‘->’ \ T c \ ‘|’ \ T c \quad \Rightarrow \ ‘->’ \\
& \quad \rightarrow B \\
\end{align*}
\]
2. Logic Programming: (25 points)

(1) Does the list \([Y, a, b, c]\) unify with the list \([a, X, Y, c]\)? If not, suggest a single change that would fix the problem. (i.e. one letter change)

(2) Consider the “family tree” example. Write a Prolog rule for a person X to be an uncle of person Y.
3. Show the Abstract Syntax Trees for the following RPAL programs: (25 points)

(1) \( f(5, 4) \) where \( h(a, b) = a - b \) within \( f(x) = h(x) \)

(2) let \( t = m \) and \( n = 2 \times m \) where \( m = 1 \) in \( t \) ls \( (n \ ls \ 1) \)

(3) let \( n(m) = \frac{m}{2} + 1 \) in \( n(m) \) where \( p = 6 \)

(4) \( m \ or \ n \rightarrow p \rightarrow m \ | \ n \ | \ n \)
4. Writing RPAL Programs: (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:

\[
\text{let } x = ((a \ast b)/c) \text{ in } ((e \text{ or } f) \rightarrow (g \rightarrow x \ast (x\ast2)\mid y) \\
\mid (f(x) \text{ where } ((f(y) = (y\ast2)) \text{ and } (x=(3 + (x \ast 2))))))
\]

Also, "beautify" the program by re-writing it on several lines with the appropriate indentation.