1. Using the RPAL string-to-tree-transduction grammar, construct the abstract syntax tree (if possible) for each of the following programs:
   (a) x or y -> x | z -> x | y
   (b) a < b -> c -> a | b | a
   (c) x aug y, z, 3
   (d) x, y, z aug 3
   (e) let f (x) = x + 2 * 1 in f (z) where z = 6
   (f) let x = z and y = 2 * z where z = 3 in x ** y ** y
   (g) let a = c in a * b ** c where ( z = 3 and y = 2 * z )
   (h) let a = c in a * b ** c where z = 3 and y = 2 * z

Answer:

(a)

```plaintext
  or
 / 
x  y
```

(b)

```plaintext
  <
   /
  a < b
    /
   c
  /
 a b
```

(c)

```plaintext
  tau
  /
 aug
 / 
 x  y
```

(d)
(e) No AST is possible.
2. Write RPAL Programs:

Write, test, and debug an RPAL program that computes the “tuple reverse” function, i.e. \( \text{Rev} (3, \text{‘hello’}, (2, 3), (\text{fn } x. x+1), \text{true}) = (\text{true}, (\text{fn } x. x+1), (2, 3), \text{‘hello’}, 3) \).

Answer:

\[
\text{let } \text{Rev } T = \text{Prev } (T, 1)
\]

\[
\text{where rec } \text{Prev } (T, N) = N \text{ gr } (\text{Order } T) \rightarrow \text{ nil}
\]

\[
| (\text{Prev } (T, N+1) \text{ aug } (T N))
\]

\[
\text{in}
\]

\[
\text{Print } (\text{Rev } (1, \text{‘2’}, (3, 2), \text{true}, (\text{fn } x.3), \text{nil}))
\]

3. Scoping Rules: (Scott Exercise 3.4 P 152)

Give three concrete examples drawn from programming languages with which you are familiar in which a variable is live but not in scope.

Answer:

(1) C: Global variable \( x \) is still live but not in the scope if a local variable \( x \) is declared in the scope.

(2) Modula-2: a global variable declared in a module is live but not in scope when execution is not inside the module.

(3) C++: non-public member variables of an object of class \( C \) are live but not in scope when execution is not inside a member function of \( C \).