Programming Languages
Sample Final Exam

Question 1
Give a regular expression for each of the following languages over \( \Sigma = \{0, 1, \ldots, 9\} \).

1. All 5-digit integers that contain no leading zeros.
2. All positive integers that begin with 9 and that are multiples of 5.
3. All strings that begin with 9 and contain three consecutive 1s.

Question 2
Give a DFA for each of the languages in Question 1.

Question 3
Give a context-free grammar for each of the following languages over \( \Sigma = \{a, b\} \).

1. \( a^* b^* \)
2. Strings that contain the same number of \( a \)'s as \( b \)'s.
3. \( \{a^n b^{n+k} | 0 \leq k\} \)

Question 4
Consider the following grammar.
\[
E \rightarrow E \text{ or } E \mid E \text{ and } E \mid \text{ not } E \mid (E) \mid x
\]
1. Prove that the grammar is ambiguous by giving an example sentence for which there are two or more parse trees.
2. Assume that the operators or and and are left-associative, the operator not is right associative, and that the operators have the precedence relation: not > and > or. Rewrite the grammar into one that does not have ambiguity and respects the associativity and the precedence of the operators.

Question 5
Write the following functions in Picat, Haskell, or Python using recursion. No higher-order functions or list comprehensions can be used in the implementations.

1. \texttt{my_zip(lst1, lst2)}: Let \( lst1 \) be \([A_1, A_2, \ldots, A_n]\), and \( lst2 \) be \([B_1, B_2, \ldots, B_n]\). This function returns the association list \([ (A_1, B_1), (A_2, B_2), \ldots, (A_n, B_n) ] \).
2. \texttt{lookup(alist, x)}: This function returns the value associated with \( x \) in the association list \( alist \). For example,
   \[
   \text{lookup}([ (’a’, 1), (’b’, 2), (’c’, 3) ], ’b’) \]
   returns 2.
3. \texttt{replicate(lst, n)}: This function replicates the elements of \( lst \) \( n \) times. For example
   \[
   \text{replicate}([’a’, ’b’, ’c’], 3) \]
Question 6

Design a data structure for binary trees, and write the following functions on binary trees in Picat, Haskell, or Python.

1. `leaves(tree)`: This function returns a list of leave values in `tree` from left to right.

2. `deepest(tree)`: This function returns the value in a deepest node in `tree`. If there are multiple such values, then the function returns the left-most one.

3. `min_max(tree)`: This function returns a pair `(min, max)`, where `min` is the minimum element, and `max` is the maximum element in `tree`. Note that the tree may not be a binary search tree.