Use only one side of the paper and start each problem on a new page!!
Please show all work to receive ANY credit!!!
1. **(12 points) Derivations and Ambiguity**

Consider the grammar for regular expressions shown below, where $E$ is the start symbol, $E$, $T$, and $F$ are non-terminals, and ‘|’, (, ), $x$, and ‘*’ are terminals.

\[
\begin{align*}
E & \rightarrow E \; | \; T \\
E & \rightarrow T \\
T & \rightarrow T \; F \\
T & \rightarrow F \\
F & \rightarrow ( \; E \; ) \\
F & \rightarrow F \; ^{\ast} \\
F & \rightarrow x
\end{align*}
\]

In the grammar, ‘|’ refers to the alternation for a regular expression, while ‘*’ refers to closure. The rule $T \rightarrow T \; F$ represents concatenation.

**a. (4 points)** Develop a leftmost derivation for the string $xx(x|x)^{\ast}$

**b. (4 points)** Develop a rightmost derivation for the string $xx(x|x)^{\ast}$

**c. (4 points)** Is the grammar ambiguous? Why or why not? Note: Be specific in your answer!!!
2. (18 points) CFGs and Left Recursion

Consider the grammar that is given below, where P is the start symbol, L and T are non-terminals, and all other symbols are terminalso

\[
\begin{align*}
A_1 & \rightarrow a A_2 a \\
A_1 & \rightarrow d A_2 d \\
A_2 & \rightarrow A_2 A_3 \\
A_2 & \rightarrow A_3 \\
A_2 & \rightarrow \epsilon \\
A_3 & \rightarrow A_2 b \\
A_3 & \rightarrow c A_2 \\
A_3 & \rightarrow A_1 b
\end{align*}
\]

Redesign the grammar given above into an equivalent form that is suitable for top-down parsing by removing any left recursion that exists. In the grammar, a, b, c and d are terminals, and A₁, A₂, and A₃ are non-terminals. Make sure that you clearly indicate your results separately, i.e., box the CFG without left recursion with \( \epsilon \) moves.
3. **(15 points total) Regular Languages, Expressions, and NFAs**
   
   For parts a and b, you may only use the operators: +, *, |, ?, (), and concatenation to construct regular expressions.

   a. **(4 points)** Write a regular expression for the language of all strings of {0, 1} whose length is a multiple of 3.

   b. **(6 points)** Construct a DFA for your answer of part a. Do not use the algorithm that converts from regular expression to an NFA to a DFA. Instead, construct your DFA directly! Clearly indicate the start state and all final states of your DFA!

   c. **(5 points)** Describe using one or two prose sentences the language that is represented by the regular expression: \((0 \mid 1)^*0(0 \mid 1)(0 \mid 1)\).
4. (15 points) FIRST and FOLLOW for CFGs

Expressions in the Lisp functional programming language are formed using nested parentheses and operators which apply to exactly two operands. Below are some example Lisp expressions with their infix equivalents:

\[
\begin{align*}
\text{(*) } x &= (+ a (* b 5)) \quad \Rightarrow \quad x = (a+(b*5)) \\
\text{(*) } y &= (- (* d (+ a 3)) (/ 8 d)) \quad \Rightarrow \quad y = (((a+3)*d)-(8/d))
\end{align*}
\]

A context-free grammar for Lisp expressions is as follows:

\[
\begin{align*}
E & \rightarrow ( = \text{id} \ L ) \\
L & \rightarrow ( A \ F \ F ) \\
L & \rightarrow ( M \ F \ F ) \\
F & \rightarrow \text{int} \mid \text{id} \mid L \\
A & \rightarrow + \mid - \\
M & \rightarrow \ast \mid \backslash
\end{align*}
\]

Note that for simplicity, \text{id} and \text{int} represent tokens for valid identifiers or integer digits. Using this grammar, compute FIRST for \text{E}, \text{L}, and \text{F}, and FOLLOW for \text{L}, \text{F}, \text{A}, and \text{M}. 

5. (15 points) Your work for the IC Graphics company and have been asked to assist in the development of a CAD/CAM mechanical design system. The system supports the design of simple mechanical parts that are constructed from two different shapes, blocks and cylinders, that may or may not contain a hole, and in turn may be combined with each other by placing shapes next to one another or on top of one another. Your responsibility at IC Graphics is to develop a lexical analyzer and parser for representing legal combinations of blocks, cylinders, and holes into simple mechanical parts. There are four different tokens, blocks/BLOCK, cylinders/CYL, blocks with holes/H-BLOCK, and cylinders with holes H-CYL. Example blocks are B1, B10, B23, B117, etc., cylinders are Ca, Cb, Cz, Cab, Ccc, etc., and holes within a block or cylinder are parenthesized, such as (B25), (Caa), (Ci), etc. Design a context free grammar that combines BLOCK, CYL, H-BLOCK, and C-BLOCK (which we’ll call objects) using two construction operations: pile-on, represented by square brackets [ ], and next-to, represented by angle brackets < >. The operation pile-on takes two or more objects and places one on top of another. The operation next-to takes two or more objects and places them adjacent to one another. For example:

\[ \text{B11} \]
\[ \text{Ca} (\text{B2}) \] yields \( \text{Ca} (\text{B2}) \)

and

\[ \text{<Cb} (\text{Cd}) \text{B6 Ch}> \] yields \( \text{Cb (Cd) B6 Ch} \)

In addition, the objects that are combined may also be the result of a pile-on or next-to construction. This leads to the following possibilities:

\[ \text{[B11 <B5 C8>]} \] yields \( \text{B11 B5 C8} \)

and

\[ \text{<[B4 B2] (C5) [C2 B1]>} \] yields \( \text{B4 C2 B2 (C5) B1} \)

The nesting can be performed to any desired level of depth. Your task is to develop a context free grammar that can represent all legal combinations of objects, using pile-on [ ] and next-to < >, to any level of nesting depth. Note that '['; ']', '<', and '>' are all tokens.