1. Given the following grammar:
   A → id = E
   E → E+T | T
   T → id

   a. Construct the DFA of LR (1) items for this grammar [10 marks]
b. Construct the LR (1) parsing table  [7 marks]

c. Trace the parser for the string a=b  [3 marks]
2. Consider the following grammar:

\[ D \to \text{var } V : T \]
\[ V \to \text{id, } V | \text{id} \]
\[ T \to \text{integer | real} \]

a. Write an attribute grammar to create entries for variables in the symbol table and insert their types [12 marks]

<table>
<thead>
<tr>
<th>Syntactic Rule</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>( D \to \text{var } V : T )</td>
<td></td>
</tr>
<tr>
<td>( V \to \text{id, } V )</td>
<td></td>
</tr>
<tr>
<td>( V \to \text{id} )</td>
<td></td>
</tr>
<tr>
<td>( T \to \text{integer} )</td>
<td></td>
</tr>
<tr>
<td>( T \to \text{real} )</td>
<td></td>
</tr>
</tbody>
</table>

b. Is your attribute grammar L-Attributed grammar or S-Attributed grammar? Explain your answer [4 marks]

c. What is the main advantage of having L-Attributed grammar? [4 marks]
3. Consider the following attribute grammar that determines the type of an expression

\[
\begin{align*}
\text{exp} & \rightarrow v \text{ exp}' & \{ \text{exp'.type} = v.\text{type} \\
& & \text{exp'.stype} = \text{exp'.type} \\
\text{exp}' & \rightarrow + \text{ exp} \text{ exp}' & \{ \text{exp'.type} = \text{check-type(exp'.type, exp.type)} \\
& & \text{exp'.stype} = \text{exp'.type} \\
\text{exp}' & \rightarrow \varepsilon & \{ \text{exp'.type} = \text{exp'.type} \\
v & \rightarrow \text{id} & \{ v.\text{type} = \text{get-type(id.name)} \} \\
v & \rightarrow \text{num} & \{ v.\text{type} = \text{integer} \}
\end{align*}
\]

Where

- \text{get-type(id.name)} is a semantic action that returns the type of the identifier if the identifier has an entry in the symbol table. Otherwise it will return invalid type.
- \text{check-type(exp', exp) checks that both types have valid and compatible types and return the appropriate type}

Draw the annotated parse tree augmented with semantic actions for the string:

\[ a+4 \] [10 marks]
4. Consider the following grammar:

\[
\begin{align*}
\text{stmt} & \rightarrow \text{if-stmt} \mid \text{ass-stmt} \mid \text{repeat-stmt} \\
\text{repeat-stmt} & \rightarrow \text{repeat stmt until (exp)} \\
\text{ass-stmt} & \rightarrow \text{var := exp} \\
\text{exp} & \rightarrow \text{exp + term} \mid \text{term} \\
\text{term} & \rightarrow \text{term > term} \mid \text{id} \mid \text{num}
\end{align*}
\]

The informal semantic of the repeat statement is to execute “\text{stmt}” until the “\text{exp}” is true. In other words the control exits from the loop when the “\text{exp}” is true.

a. Write an attribute grammar for generating three-address code for the following rules from the above grammar. [15 marks]

<table>
<thead>
<tr>
<th>Syntactic Rule</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeat-stmt \rightarrow \text{repeat stmt until (exp)}</td>
<td></td>
</tr>
<tr>
<td>ass-stmt \rightarrow \text{var := exp}</td>
<td></td>
</tr>
<tr>
<td>exp \rightarrow \text{exp + term}</td>
<td></td>
</tr>
<tr>
<td>exp \rightarrow \text{term}</td>
<td></td>
</tr>
<tr>
<td>term \rightarrow \text{term &gt; term}</td>
<td></td>
</tr>
<tr>
<td>term \rightarrow \text{id}</td>
<td></td>
</tr>
<tr>
<td>term \rightarrow \text{num}</td>
<td></td>
</tr>
</tbody>
</table>
b. Write the three address code generated for the statement:  

\[
\text{S:= 0;}
\]
\[
\text{repeat S=S+1 until (S>100);} 
\]
5. Consider the following grammar:

\[
\begin{align*}
ass & \rightarrow v = exp \\
exp & \rightarrow exp + term | term \\
term & \rightarrow v | num | num. num \\
v & \rightarrow id
\end{align*}
\]

Write an attribute grammar to generate three-address code for the assignment statement using the above grammar according to the following rules:

a. In case of integer operands the generated operator is ADDI.

b. In case of float operands the operator generated is ADDF.

c. In case of mixed types a three address code must be generated to convert the integer operand to float by using the unary operator FLOAT and then generate the ADDF instruction.

d. In case of assignment statement with mixed type a three address code must be generated to convert the result of the expression on the RHS to float by using the unary operator FLOAT and then generate the assignment instruction.

(Hint: Assume that the types have been already checked and the type attribute takes only integer or float) [10 marks]

<table>
<thead>
<tr>
<th>Syntactic Rule</th>
<th>Semantic Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ass \rightarrow v = exp)</td>
<td></td>
</tr>
<tr>
<td>(exp \rightarrow exp + term)</td>
<td></td>
</tr>
<tr>
<td>(exp \rightarrow term)</td>
<td></td>
</tr>
<tr>
<td>(term \rightarrow v)</td>
<td></td>
</tr>
<tr>
<td>(term \rightarrow num)</td>
<td></td>
</tr>
<tr>
<td>(term \rightarrow num. num)</td>
<td></td>
</tr>
<tr>
<td>(v \rightarrow id)</td>
<td></td>
</tr>
</tbody>
</table>

[Note: Try to stick to the indicated space for answering the question]