EXAM 2 - 447 COMPILER DESIGN

1. (5 marks) Consider the following grammar:

   decl → type var-list
   type → int | float
   var-list → var-list ,id | id

   a. Construct the DFA of LR(1) items for this grammar

   b. Construct the LR(1) parsing table
2. (a) (5 marks) For the input string A,B,C : REAL, construct an annotated parse tree according to the following syntax directed definition:

\[
\begin{align*}
F & \rightarrow \text{id}, S \quad \{ \text{S.entry := id.entry} \} \\
S & \rightarrow \text{id}, S_1 \quad \{ \text{addlink (S.entry, id.entry)} \\
    & \quad \text{S}_1.\text{entry := id.entry} \}
S & \rightarrow \text{id} : T \quad \{ \text{addlink (S.entry, id.entry)} \\
    & \quad \text{addtype (id.entry, T.type)} \}
T & \rightarrow \text{INTEGER} \quad \{ \text{T.type := integer } \}
T & \rightarrow \text{REAL} \quad \{ \text{T.type := real } \}
\end{align*}
\]

b) (3 marks) For the attributes of the syntax directed definition of (1), indicate whether each attribute is inherited or synthesized using the following table:

<table>
<thead>
<tr>
<th></th>
<th>inherited</th>
<th>synthesized</th>
</tr>
</thead>
<tbody>
<tr>
<td>id.entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.entry</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T.type</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3. Given the following grammar:

\[
\begin{align*}
S \rightarrow & \text{ if } B \text{ then } S \text{ else } S \\
S \rightarrow & \text{ while } B \text{ do } S \\
S \rightarrow & \text{ id } := E \\
B \rightarrow & \text{ id } \text{ relop } \text{ id} \\
B \rightarrow & B \text{ and } B \\
B \rightarrow & \text{ not } B \\
E \rightarrow & E + E \\
E \rightarrow & \text{ id} \\
E \rightarrow & \text{ id } (E)
\end{align*}
\]

a)(8 marks) define a syntax directed translation scheme to produce three address code.
b)(4 marks) Generate the three address code for the statement:

```plaintext
while a<b and not c=d do
    if x<y then
        z := w(a) + b
    else z := w(a) + z;
    a := a + 1
```

4. (5 marks) The following grammar defines an expression that does not accept any mixed type operation. For integer and real types, the '+' operators stand for arithmetic addition whereas it stands for logical 'OR' and when the operators are of boolean type. If the expression is of type character the '+' operator stands for concatenation.

```
E ---> E+E | id | num | num.num | TRUE | FALSE | string
```

Write translation rules to determine the types of expression E.