Exercise 1 Ambiguous grammars

a) Consider the following grammar (given in Java CUP Syntax):

```java
expr ::= 
    expr LESS expr 
  | expr AND expr 
  | NUMBER 
  | ID 

; 
```

Show that this grammar is ambiguous.

b) To fix the ambiguity in the grammar your colleague suggests the following grammar:

```java
expr ::= 
    atomicExpr operator expr 
  | atomicExpr 

; 

atomicExpr ::= 
    NUMBER 
  | ID 

; 

operator ::= 
    LESS 
  | AND 

; 
```

What is the problem with this solution? Can you give a better solution for the problem?
Exercise 2 Towards LR(0)-DFA construction

Let Γ be a grammar with start symbol $S$ and productions:

$$
S \rightarrow TV \\
T \rightarrow i \mid (TxT) \\
V \rightarrow i \mid n \mid (VIV)
$$

a) Give an execution of an LR parser accepting the input $(i \times i) ((i\ i\ n)\ i\ i)$.

(Complete the table below the parser state for each step)

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Next Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$(i \times i) ((i\ i\ n)\ i\ i)$</td>
<td>shift</td>
</tr>
<tr>
<td></td>
<td>$(i \times i) ((i\ i\ n)\ i\ i)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i</td>
<td></td>
</tr>
</tbody>
</table>

b) How did you decide whether to reduce using $T \rightarrow i$ or $V \rightarrow i$?

c) How did you decide whether to shift $i$ or reduce using $V \rightarrow i$?

d) Describe how one can pick the next action by just looking at the current stack.

e) Give a regular expression describing the state of the stack, such that the regular expression matches the stack iff the next action should be “reduce using $V \rightarrow i$”.

Exercise 3 LR(0)-DFA construction

a) Construct the LR(0)-DFA for the grammar from exercise 2.

b) Construct the LR(0)-DFA for the following grammar (given in Java CUP Syntax).

```java
// this first rule would not be written down in CUP
Start ::= statements EOF;

statements ::= 
    stmt statements 
    | /* empty */ / 
    ;

stmt ::= 
    type ID SEMI 
    | expr SEMI 
    | expr EQ expr SEMI 
    ;

type ::= ID;

expr ::= 
    ID 
    | expr DOT ID 
    ;
```