FILS
Compiler Techniques

Review

A. Theoretical questions (30 min)

1. What is the structure of a compiler? Explain the components and their interactions. (5 min)
2. Define Knuth's attribute grammar. (5 min)
3. Give the definition of LL(1) grammars. (5 min)
4. Give the parser classification. Explain characteristics of each parser type in terms of underlying machines. (5 min)
5. What is an ambiguous grammar? Give an example. (5 min)
6. Define the Syntax-Directed Translation scheme. Give a short example. (5 min)

B. Exercises

1. Consider the following grammar of some statements in programming languages:

   S -> **if** E **then** S
   S -> **if** E **then** S **else** S
   S -> **while** E **do** S
   S -> **begin** L **end**
   S -> E
   L -> L **;** S
   L -> S
   E -> E **oprel** E
   E -> **id**

   a. Transform it in a LL(1) grammar
   b. Build the action table of a descendent parser of the transformed grammar. (20 min)

2. Consider the following grammar of postfix expressions:

   E -> EE**+**
   E -> EE*****
   E -> **a**

   a. Augment it and build the SLR sets of items and their GOTO function.
   b. Indicate any action conflicts in your sets of items.
   c. Construct the SLR-parsing table, if one exists. If it does not verify if a LALR or LR(1) table exists. (20 min)

3. Consider the following attribute-grammar:

$P \rightarrow E\ \$\$$	$\triangleright$ E.d = 0	$\triangleright$ P.m = E.m	
$E \rightarrow A$	$\triangleright$ A.d = E.d	$\triangleright$ E.m = A.m	
$E \rightarrow B$	$\triangleright$ B.d = E.d	$\triangleright$ E.m = B.m	
$E \rightarrow \epsilon$	$\triangleright$ E.m = E.d		
$A \rightarrow (\ E_1\ )\ E_2$	$\triangleright$ E_1.d = A.d + 1	$\triangleright$ E_2.d = A.d	$\triangleright$ A.m = max(E_1.m, E_2.m)
$B \rightarrow [\ E_1\ ]\ E_2$	$\triangleright$ E_1.d = B.d + 1	$\triangleright$ E_2.d = B.d	$\triangleright$ B.m = max(E_1.m, E_2.m)

   a) Draw the parse tree for the string: [()]() $$.
   b) List the attributes of the grammar symbols, saying which is synthesized and which is inherited.

c) Is the grammar S-attributed? Is it L-attributed? Explain. (10 min)

4. Let consider a grammar for expressions involving operator + and integer or floating-point operands. Floating-point numbers are distinguished by having a decimal point:

   E -> E **+** T
   E -> T
   T -> **num**.**num**
   T -> **num**

   a) Give an SDD to determine the type of each term T and expression E.

   b) Extend your SDD to a SDT schema of (a) to translate expressions into postfix notation. Use the unary operator **intToFloat** to turn an integer into an equivalent float. (15 min)

5. Generate an optimized three-address intermediate code for the following statements: (15 min)

```
while x<10||x>20 && x< y do {
        a [x] = y*x;
        x++;
}
x=y+5;
```