# Programming Languages / Programming Language Technology

Exam, 9 March 2009

8.30–12.30 in H

Course codes: Chalmers TIN321, GU DIT229 (BSc); Chalmers DAT150, GU DIT230 (MSc)

Teacher: Aarne Ranta (tel. 1082)

**Grading scale**: Max = 60p, BSc (TIN321, DIT229): VG = 5 = 48p, 4 = 36p, G = 3 = 24p.

**Aids**: an English dictionary.

**Instructions**

This exam has three groups of questions, one **easy**, one **intermediate**, and one **advanced**. Points are distributed in such a way that doing the easy questions is enough to pass the exam (mark 3 or G). All the easy and around half of the intermediate are needed to get mark 4. All the intermediate, or half of them and the advanced, are needed to get the highest mark (5 or VG).

You can get points for the more difficult parts without doing the easier ones.

The bonus points from exercises are added to the total score by the teachers.

Questions requiring answers in code can be answered in any of: C, C++, Haskell, Java, or precise pseudocode.

Text in the answers can be in any of: Bulgarian, Danish, Dutch, English, Estonian, Finnish, French, German, Italian, Norwegian, Polish, Russian, Spanish, and Swedish.

For any of the six questions, an answer of less than one page should be enough.

**Group 1: easy questions**

1. Write a BNF grammar that covers the following kinds of constructs in C++:

- `if` statements with and without `else`
- `return` statements with an expression
- expressions: identifiers, boolean literals `true` and `false`, conjunctions (`a && b`), and disjunctions (`a || b`)

The expressions have their usual precedences; in particular, the conjunction binds stronger than the disjunction, and these are both left-associative. An example statement is shown in question 2. You can use the standard BNFC categories `Integer` and `Ident`, as well as the `coercions` macro. (10p)

2. Show the parse tree and the abstract syntax tree of the statement

```
if (true) return a || b && c ; else return false ;
```

in the grammar that you wrote in question 2. (5p for each tree)

3. Write syntax-directed type checking code (or pseudocode) for `if` statements with and without else, as well as for lazy conjunctions (`&&`) and disjunctions (`||`). (5p)

Write syntax-directed interpreter code (or pseudocode) for lazy conjunctions (`&&`) and disjunctions (`||`). Expressions can have side effects! (5p)

**Group 2: intermediate questions**

4. Trace the LR parsing process of the statement in Question 2 above, according to your own grammar from Question 1. That is, show all actions and the resulting stack and the state. (7p)

Is your grammar in Question 1 LL(1)? If not, explain why. (3p)

5. Write the compilation schemes for boolean literals `true` and `false`, and for lazy conjunctions and disjunctions (`a && b`, `a || b`) in JVM (using Jasmin assembler). Explain each of the instructions occurring in the schemes by giving their small-step semantics. It is not necessary to remember exactly the names of the instructions - only what arguments they take and how they work. (5p for the schemes, 5p for the semantics)

**Group 3: advanced questions**

6. Write the big-step operational semantic rules for call-by-value lambda calculus, covering abstractions, applications, identifiers, and integer constants. You can use either closures (6p) or ordinary substitutions (4p).

Show the evaluation tree of the expression `(\x -> x) (\x -> x) 6` by using these rules. In other words, show a derivation of the judgement where this expression gets its value in the empty environment. (4p).