

Programming Language Technology

Exam, 15 January 2026 at 8.30 – 12.30 in HB 1-4

Course codes: Chalmers DAT151, GU DIT231.

Exam supervision: Andreas Abel (+46 31 772 1731), visits at 9:30 and 11:30.

Grading scale: Max = 60p, VG = 5 = 48p, 4 = 36p, G = 3 = 24p.

Allowed aid: an English dictionary.

Exam review: 28 January 2026 14.30-15.30 in EDIT meeting room 6128 (6th floor).

Please answer the questions in English.

Question 1 (Grammars): Write a labelled BNF grammar that covers the following kinds of constructs of C/C++/Java (sublanguage of lab 2):

- Program: a sequence of function definitions.
- Function definition: type followed by identifier, comma-separated parameter list in parentheses, and block.
- Parameter: type followed by identifier, e.g. `int x`.
- Block: a sequence of statements enclosed between `{` and `}`
- Statements:
 - block
 - initializing variable declaration, e.g., `int x = 42;`
 - expression followed by semicolon
 - while statement
- Expressions, from highest to lowest precedence:
 - atoms: identifier, integer literal, function call
 - addition (`+`), left associative
 - less-or-equal-than `integer` comparison (`<=`), non-associative
 - assignment, right associative

Putting an expression into parentheses makes it an atom.

- Type: `void` or `int` or `bool`

Line comments are started by `#`. You can use the standard BNFC categories `Integer` and `Ident` and any of the BNFC pragmas (`coercions`, `terminator`, `separator` ...). An example program is:

```
#include <stdio.h>
#define printInt(x) printf("%d\n",x)
void fib(int n) {
    int i = 0; int cur = 0; int next = 1;
    while ((i = i + 1) <= n) { int tmp = next; next = tmp + cur; cur = tmp; }
    printInt(cur);
}
```

(10p)

Question 2 (Compilation Phases):

1. List the names of the phases of a compiler in the order they execute in the lab 3 compiler. (1p)
2. Each of the following snippets (from the C– language of lab 2) contains one error. For each of the snippets, write the name of the compilation phase in which this error ought to be detected and reported. (4p)
 - (a) `while ((i == i + 1) <= n) { int t = x; x = t + c; c = t; }`
 - (b) `while ((i = i + 1) <= n) { int t = x; x = t + ; c = t; }`
 - (c) `while ((i = i + 1) <= n) { int t = x; x = t + c; int t = 0; }`
 - (d) `while ((i = i "+ 1) <= n) { int t = x; x = t + c; c = t; }`

CLARIFICATION: The variables `i`, `n`, `x` and `c` are assumed to be in scope and of type `int`.

Question 3 (LR Parsing):

Consider the following labeled BNF-Grammar.

```
Sub.    E ::= E "-" E ;
One.   E ::= "1" ;
Two.   E ::= "2" ;
Three. E ::= "3" ;
```

We work with the following example string:

`3 - 2 - 1`

1. The grammar is ambiguous. Show this by giving two different parse trees for the example string.
2. The LR parser generated for this grammar has a shift-reduce conflict.

Step by step, trace the parsing of the example string showing how the stack and the input evolve and which actions are performed. Resolve any shift-reduce conflict in favor of *shift*.

Which of the two parse trees is produced by this run?

3. Now do the trace again, this time resolving conflicts in favor of *reduce*.

Which of the two parse trees is produced by that run?

(8p)

Question 4 (Type checking and evaluation):

1. Write syntax-directed typing rules for the *expressions* of Question 1. Alternatively, you can write the type-checker in pseudo code or Haskell. Functions `lookupVar` and `lookupFun` can be assumed. In any case, the typing environment must be made explicit. (6p)
2. Write syntax-directed interpretation rules for the *statements*, *blocks* and *statement sequences* of Question 1, assuming an interpreter for expressions $\gamma \vdash e \Downarrow \langle v; \gamma' \rangle$. Alternatively, you can write the interpreter in pseudo code or Haskell. Functions `evalExp` and `lookupVar` can be assumed. In any case, the environment must be made explicit. (7p)

Question 5 (Compilation):

1. *Statement by statement*, translate the example program of Question 1 to Jasmin. (Do not optimize the program before translation!)

It is not necessary to remember exactly the names of the JVM instructions—only what arguments they take and how they work.

Make clear which instructions come from which statement, and determine the stack and local variable limits. (8p)

2. Give the small-step semantics of the JVM instructions you used in the Jasmin code in part 1 (except for function call and `return` instructions). Write the semantics in the form

$$i : (P, V, S) \longrightarrow (P', V', S')$$

where (P, V, S) is the program counter, variable store, and stack before execution of instruction i , and (P', V', S') are the respective values after the execution. For adjusting the program counter, you can assume that each instruction has size 1. (6p)

Question 6 (Functional languages):

1. The following grammar describes a tiny simply-typed sub-language of Haskell.

x		identifier
i	$::= 0 \mid 1 \mid -1 \mid 2 \mid -2 \mid \dots$	integer literal
e	$::= i \mid e + e \mid x \mid \lambda x \rightarrow e \mid e e$	expression
t	$::= \text{Int} \mid t \rightarrow t$	type

Application $e_1 e_2$ is left-associative, the arrow $t_1 \rightarrow t_2$ is right-associative. Application binds strongest, then addition, then λ -abstraction.

For the following typing judgements $\Gamma \vdash e : t$, decide whether they are valid or not. Your answer can be just “valid” or “not valid”, but you may also provide a justification why some judgement is valid or invalid.

- (a) $a : \text{Int}, b : \text{Int} \rightarrow \text{Int} \quad \vdash (\lambda c \rightarrow 0 + b) a : \text{Int}$
- (b) $a : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int} \quad \vdash \lambda b \rightarrow b (a b) : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$
- (c) $a : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int} \quad \vdash \lambda b \rightarrow a (a b) : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int}$
- (d) $a : \text{Int} \rightarrow \text{Int} \quad \vdash \lambda b \rightarrow a (a b + a 0) : \text{Int} \rightarrow \text{Int}$
- (e) $a : (\text{Int} \rightarrow \text{Int}) \rightarrow (\text{Int} \rightarrow \text{Int}) \vdash (\lambda b \rightarrow a b) (\lambda a \rightarrow 1) : \text{Int} \rightarrow \text{Int}$

The usual rules for multiple-choice questions apply: For a correct answer you get 1 point for a wrong answer -1 points. If you choose not to give an answer for a judgement, you get 0 points for that judgement. Your final score will be between 0 and 5 points, a negative sum is rounded up to 0. (5p)

2. For each of the following terms, decide whether it evaluates more efficiently (in the sense of fewer reductions) in call-by-name or call-by-value. Your answer can be just “call-by-name” or “call-by-value”, but you can also add a justification why you think so. *Same rules for multiple choice as in part 1. (5p)*

- (a) $(\lambda x \rightarrow \lambda y \rightarrow x + x + x) (1 + 2 + 3) (4 + 5 + 6 + 7)$
- (b) $(\lambda x \rightarrow \lambda y \rightarrow x + x + x) (1 + 2) (3 + 4 + 5 + 6)$
- (c) $(\lambda x \rightarrow x) ((\lambda y \rightarrow \lambda z \rightarrow z + z + z) (1 + 2 + 3 + 4) (5 + 6))$
- (d) $(\lambda x \rightarrow \lambda y \rightarrow y + y) ((\lambda z \rightarrow z z) (\lambda z \rightarrow z z)) (1 + 2 + 3)$
- (e) $(\lambda x \rightarrow \lambda y \rightarrow y + y + y) (\lambda u \rightarrow (\lambda z \rightarrow z z) (\lambda z \rightarrow z z)) (1 + 2)$

Good luck!