

Tentamen i

KOMPILATORTEKNIK

Kurskod:	SMD163
Datum:	2004 – 03 – 17
Skrivtid:	5 timmar
Totalt antal uppgifter:	8
Totalt antal poäng:	40
Jourhavande lärare:	Johan Nordlander
Telefon:	070 – 620 74 61
Tillåtna hjälpmedel:	Engelskt lexikon

1. (5 marks)

- a) Describe the different formats a program takes as it is processed by the different phases of a typical modern compiler.
- b) Name two phases that are usually implemented as a single compiler *pass*.
- c) What could be the reason for implementing a whole compiler as one single pass?
- d) Give an example of a feature in Java that makes a single-pass implementation extremely complicated (if not impossible).

2. (5 marks)

- a) In the context of compilers, what is the common application of *deterministic finite automata* (DFAs)?
- b) Describe the difference between deterministic and a *non-deterministic finite automata* (NFAs).
- c) What is the motivation for studying non-deterministic automata at all?
- d) Discuss if there is, or if there ought to be, a similar distinction between deterministic and non-deterministic *parsers* for context-free languages.

3. (5 marks)

Consider the following grammar for a fictitious imperative language *L*:

```
Prog ::= Decl ';' Prog
      | Call
Decl ::= Type ID '(' Par ')' Stmt 'return' Exp
Par  ::= Type ID
      | Par ';' Par *
Type ::= 'int' | 'bool'
Stmt ::= ID '=' Exp
      | 'while' '(' Exp ')' Stmt *
      | Stmt ';' Stmt *
Exp  ::= ID | INTEGER | 'true' | 'false'
      | Exp Op Exp *
      | Call
Op   ::= '+' | '*' | '<' | '&'
Call ::= ID '(' Arg ')'
Arg  ::= Exp
      | Arg ';' Arg *
```

Strings in quotes are considered to be terminal symbols, in conjunction with the all-uppercase names ID and INTEGER.

The productions marked with asterisks are all sources of shift/reduce conflicts when the grammar is processed by an LALR parser generator tool like Yacc/Jacc. Discuss for each case what the conflict means in terms of parse-trees alternatives, and which of the following responses you find most appropriate:

- Relying on the default parser behavior that favors shift over reduce.
- Adding precedence directives (show which ones).
- Rewriting the grammar while keeping the syntax intact (describe how).
- Redefining the concrete syntax (describe how).

You may assume that all symbols and operators have their standard interpretation (with single '&' meaning logical *and*), and that established precedences and associativities should apply.

4. (5 marks)

Construct an LR(0) parser in the form of a state diagram for the following subset of the L grammar:

$$\begin{array}{l} \text{Exp} ::= \text{ID} \\ \quad | \text{Exp Op Exp} \\ \text{Op} ::= '+' \\ \quad | '**' \end{array}$$

Include the full set of items in each state, and mark all transitions with their respective symbols. Clearly indicate if there are conflicts in any state, and discuss whether using lookahead symbols (as in SLR(1), LALR(1) or (LR(1)) would resolve them.

5. (6 marks)

Below follows an excerpt from a formal type system definition for the language L .

$$\frac{E(v) = t \quad E \vdash e : t}{E \vdash v = e \text{ well-typed}} \quad \frac{E \vdash e : \text{bool} \quad E \vdash s \text{ well-typed}}{E \vdash \text{while}(e) s \text{ well-typed}}$$

$$\frac{E(v) = t}{E \vdash v : t} \quad \frac{E \vdash e : \text{int} \quad E \vdash e' : \text{int}}{E \vdash e + e' : \text{int}} \quad E \vdash n : \text{int}$$

$$\frac{E(f) = t(t_1 \dots t_n) \quad E \vdash e_1 : t_1 \dots E \vdash e_n : t_n}{E \vdash f(e_1 \dots e_n) : t} \quad \frac{E \vdash e : \text{int} \quad E \vdash e' : \text{int}}{E \vdash e < e' : \text{bool}}$$

Here the meta symbol e stands for an expression, s stands for a statement, while $t, v,$ and n range over types, variables and integer constants, respectively.

- Discuss informally what types you would like to see for g and i if the statement $\text{while}(g(i)) i = i + 2$ is going to be free from type-errors.
- Use the rules above, construct a formal proof that the statement $\text{while}(g(i)) i = i + 2$ is well-typed in an environment E that meets your criteria from a).

- c) Describe how a compiler based on these rules would detect the type-error in statement $while (g(i)) i = i < 2$ (assuming the same environment as in b)).

6. (4 marks)

- a) Show how a simple compilation scheme for the language L might generate IA32 assembly code for a function call of the form $f(e_1, e_2, e_3)$.
- b) Assuming that f is defined as $int f(int x, int y, int z) \{ return z; \}$, show the assembly code you would generate for this function, expressed in terms of some unspecified statement compilation scheme S .

7. (4 marks)

Define the term *peephole optimization*, give an example of such an optimization using concrete assembly code, and sketch briefly how a peephole optimization pass could be implemented in a Java-based compiler.

8. (6 marks)

Consider extending the language L with

- a) Global variables.
- b) Local variables.
- c) Java-style arrays with garbage-collection.

Describe how implementing each of these extensions will affect the following data structures of an L compiler:

- 1) Abstract syntax.
- 2) Environment (symbol table).
- 3) Representation of types used in the type-checker.
- 4) Run-time memory layout of the compiled program.

Note that you are essentially given 12 questions in one here! Answer each of them briefly, though, staying well within a few lines of text for each item.