Linköpings universitet
IDA – Department of Computer and Information Science
Professor Peter Fritzson

TENTAMEN / *EXAM*

TDDD16 Kompilatorer och interpretatorer / *Compilers and Interpreters*
TDDB44 Kompilatorkonstruktion / *Compiler Construction*

April 7, 2010, 8:00–12:00, T2 and U1

Jour: Kristian Stavåker, 0763-361782 and 013-284093; (Will come approx 9.00 and 10.30)

Hjälpmedel / *Allowed material:*

- Engelsk ordbok / Dictionary from/to English to/from your native language;
- Miniräknare / Pocket calculator

General Instructions

- This exam has 10 assignments and 5 pages, including this one.
- Read all assignments carefully and completely before you begin.
- The first assignment (on formal languages and automata theory) is ONLY for TDDD16, while the last one (on code generation for RISC, etc.) is ONLY for TDDB44.
- It is recommended that you use a new sheet for each assignment. Number all your sheets, and mark each sheet on top with your name, personal number/personnummer, and the course code.
- You may answer in either English or Swedish.
- Write clearly. Unreadable text will be ignored.
- Be precise in your statements. Unprecise formulations may lead to a reduction of points.
- Motivate clearly all statements and reasoning.
- Explain calculations and solution procedures.
- The assignments are *not* ordered according to difficulty.
- The exam is designed for 40 points (per course). You may thus plan about 6 minutes per point.
- Grading: U, 3, 4, 5.
- For exchange students (with a .in the personnummer) ECTS marks will be applied.
- The preliminary threshold for passing (grade 3) is 20 points.

1. Only TDDD16: (6p) Formal Languages and Automata Theory

Consider the language L consisting of all strings w over the alphabet {c,y} such that w contains yy or cccc (i.e., at least 2 y:s in sequence, or at least 4 c:s in sequence, or both). For example, the strings cycyy, ccccyc, yyccccy, yy, etc. belong to the language L.

(a) Construct a regular expression for L (1.5p)

(b) Construct from the regular expression an NFA recognizing L (1.5p)

(c) Construct a DFA recognizing L, either by deriving from the NFA of question (1b), or by constructing one directly. (2.5p)

(d) Give an example of a formal language that is not context-free. (0.5p)

2. (3p) Compiler Structure and Generators

(a) What are the generated compiler phases and corresponding specification formalisms (mention at least 5) when using a compiler generator to generate a compiler? (2p)

(b) Most modern compilers have not just one but several intermediate representations.

i. Explain how these are, in general, related to each other, and what this organization means for the code generation process. (1p)

3. (3p) Error Handling

Explain, define, and give examples of using the following concepts regarding error handling:

a) Valid prefix property. (1p)
b) Phrase level recovery. (1p)
c) What is language independent error handling? Mention at least one such tool. (1p)

4. (6p) Top-Down Parsing

Given a grammar with nonterminals <S>, <X>, <Y>, <Z> where <S> is the start symbol and the following productions

```
1. <S> ::= <X> $ <X>
2. <X> ::= <Y> % <X>
3.       |  <Y>
4. <Y> ::= <Y> & <Z>
5.       |  <Z>
6. <Z> ::= a
7.       |  b
```

Assume that `$`, `%` and `&` are operators.

(a) What is the associativity (left, right, or none) of the operators? (1p)

(b) What is the precedence (relative priority) between the operators? (1p)

(c) Can the given grammar be used directly for a recursive-descent parser? Motivate your answer. If not, rewrite the grammar so that the language generated from it can be analyzed using the recursive-desent method. (2p)

(d) Write a recursive-descent parser to analyze the language defined by the grammar. (2p)

5. (4p) LR Parsing

(a) (3p) Use the SLR(1)-tables below to show how the following sentence is parsed:

```
a & b $ a % b
```

You should show, step by step, how stack, input data, etc., are changed during the parsing. Note that _!_ is a terminal symbol for end-of-file. Start state is 0, <S> is the start symbol.

Grammar rules:

```
1. <S> ::= <X> $ <X>
2. <X> ::= <Y> % <X>
3.       |  <Y>
4. <Y> ::= <Y> & <Z>
5.       |  <Z>
6. <Z> ::= a
7.       |  b
```

Tables:

	Action						Goto			
State	_!_	$	%	&	a	b	<S>	<X>	<Y>	<Z>
0	*	*	*	*	S9	S10	1	2	5	8
1	A	*	*	*	*	*	*	*	*	*
2	*	S3	*	*	*	*	*	*	*	*
3	*	*	*	*	S9	S10	*	4	5	8
4	R1	*	*	*	*	*	*	*	*	*
5	R3	R3	S6	S11	*	*	*	*	*	*
6	*	*	*	*	S9	S10	*	7	5	8
7	R2	R2	*	*	*	*	*	*	*	*
8	R5	R5	R5	R5	*	*	*	*	*	*
9	R6	R6	R6	R6	*	*	*	*	*	*
10	R7	R7	R7	R7	*	*	*	*	*	*
11	*	*	*	*	S9	S10	*	*	*	12
12	R4	R4	R4	R4	*	*	*	*	*	*

(b) (1p) Draw an abstract syntax tree for the sentence that was parsed in a).

6. (3 p) Symbol Table Management

The C language allows static nesting of scopes for identifiers, determined by blocks enclosed in braces.

Given the following C program:

```
int pelle;
int main( void )
{
  int kalle, pelle;
  // ... some statements omitted
  if (kalle==0) {
    int nisse,pelle;
    // ... some statements omitted
    for (nisse=0; nisse<100; nisse++) {
      double kalle,nisse;
      // ... some statements omitted
      kalle = pelle * 2;
    }
  }
```

```
}
```

For the program point containing the assignment `kalle = pelle * 2`, show how the program variables are stored in the symbol table if the symbol table is to be realized as a hash table with chaining and block scope control. Assume that your hash function yields value 3 for `kalle`, value 2 for `nisse` and `pelle`, and value 5 for `main`. (2p)

Show and explain how the right entry of the symbol table will be accessed when looking up identifier pelle in the assignment `kalle = pelle * 2`. (0.5p)

When generating code for a block, one needs to allocate run-time space for all variables defined in the block. Given a hash table with chaining and block scope control as above, show how to enumerate all variables defined in the current block, without searching through the entire table. (0.5p)

7. (6 p) Syntax-directed translation

An Algol-like language is augmented with an `if2`-statement in the following way:

```
<if2_statement> ::= if2(<expression_1>,<expression_2>)
none: <statement_1>
first: <statement_2>
secnd: <statement_3>
both: <statement_4>
endif2;
```

The `if2`-statement works like the following nesting of if statements:

```
if <expression_1>
  then if <expression_2>
    then <statement_4>
    else <statement_2>
  else if <expression_2>
    then <statement_3>
    else <statement_1>;
```

Write the semantic rules - a syntax directed translation scheme - for translating the `if2`-statement to quadruples. Assume that the translation scheme is to be used in a bottom-up parsing environment using a semantic stack. Use the grammar rule above as a starting point, but maybe it has to be changed.

You are not allowed to define and use symbolic labels, i.e. all jumps should have absolute quadruple addresses as their destinations. Explain all the attributes, functions, and instructions that you introduce. State all your assumptions.

8. (7p) Intermediate Code Generation and Optimization

Given the following code segment:

```
for k:=1 to 35 do
if k>w+5
then v:=(v+3)/6
else y:=y-1;
```

(a) Translate the code segment into abstract syntax trees, quadruples, and postfix code. (3 p)

(b) Divide the following code segment into basic blocks, draw a control flow graph, and show as well as motivate the existing loop/loops (3 p)

```
1: x := 10;
2: x := x + 1
```

```
3: i := x + 3;
4: if i <= 50 goto 7
5: i := i - x
6: goto 8
7: i := i + x;
8: m := n + 3;
9: i :=i + 1;
10: if x < 10 goto 2;
```

(c) Mention a code transformation that can be done on loops, and perform it on the above example by help of the control flow graph (1 p)

9. (2 p) Memory management

What property of programming languages requires the static link in the procedure's activation record? What is the purpose of the static link, i.e., how and when is it used?

What is a dynamic link and how is it used?

10. Only TDDB44: (6 p) Code Generation for RISC ...

(a) A *structural hazard* in a pipelined processor is a resource conflict where several instructions compete for the same hardware resource in the same clock cycle.

i. How do superscalar processors handle structural hazards? (0.5p)

ii. Certain processor architectures leave it to the assembler-level programmer or compiler to make sure that structural hazards do not occur. Sketch the technique (data structure, principle) that can be used in instruction scheduling to avoid structural hazards. (1p)

(b) Given the following medium-level intermediate representation of a program fragment (derived from a `while` loop):

```
1: s = 3
2: p = 4.1
3: goto 9
4: a = s / 3
5: b = a + p
6: s = a - b
7: d = p
8: p = p * 0.7
9: f = (p > 0.33)
10: if f goto 4
11: d = d / b
```

Identify the live ranges of program variables, and draw the live range interference graph

(i) for the loop body in lines 4–9,

(ii) for the entire fragment.

For both (i) and (ii), assign registers to all live ranges by coloring the live range interference graph. How many registers do you need at least, and why? (3.5p)

(c) What is the basic idea of *software pipelining* of loops? (1p)

Good luck!