Försättsblad till skriftlig tentamen vid Linköpings universitet

(fylls i av ansvarig)

Datum för tentamen	2010-08-23
Sal	TER2
Tid	14-18
Kurskod	TDDB44
Provkod	TEN1
Kursnamn/benämning	Kompilatorkonstr.
Institution	IDA
Antal uppgifter som ingår i tentamen	10
Antal sidor på tentamen (inkl. försättsbladet)	6
Jour/Kursansvarig	K. Stavåker P. Fritzson
Telefon under skrivtid	0763-361782, 0708-281484
Besöker salen ca kl.	
Kursadministratör (namn + tfnnr + mailadress)	G. Mellheden 2297 gunme@ida.liu.se
Tillåtna hjälpmedel	Ordbok Miniräknare
Övrigt (exempel när resultat kan ses på webben, betygsgränser, visning, övriga salar tentan går i m.m.)	
Vilken typ av papper ska användas, rutigt eller linjerat	
Antal exemplar i påsen	

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*

August 23, 2010, 14:00–18:00, TER2

Jour: Kristian Stavåker, 0763-361782 or Peter Fritzson 0708-281484; (Will come approx 15.15)

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 {d,u} such that w contains uu or ddddd (i.e., at least 2 u:s in sequence, or at least 5 d:s in sequence, or both). For example, the strings duduu, dddddud, uudddddu, uu, 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.

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>`, `<A>`, `<B>`, `<C>` where `<S>` is the start symbol and the following productions

```
1. <S> ::= <A> $ <A>
2. <A> ::= <B> # <A>
3.       | <B>
4. <B> ::= <B> & <C>
5.       | <C>
6. <C> ::= 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 ida;
int main( void )
{
  int sara, ida;
  // ... some statements omitted
  if (sara==0) {
    int hannah,ida;
    // ... some statements omitted
    for (hannah=0; hannah<100; hannah++) {
      double sara,hannah;
      // ... some statements omitted
      sara = ida * 3;
    }
  }
```

```
}
```

For the program point containing the assignment `sara = ida * 3`, 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 `sara`, value 2 for `hannah` and `ida`, and value 5 for `main`. (2p)

Show and explain how the right entry of the symbol table will be accessed when looking up identifier `ida` in the assignment `sara = ida * 3`. (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. (5 p) Syntax-directed translation

A Pascal-like language is extended with a `restartblock` statement according to the following grammar:

```
<block>     ::= begin <stmt_list> end
<stmt_list> ::= <stmt_list> <stmt> |
<stmt>      ::= <assignment> | ... | restartblock
```

(where "..." represents all other possible kinds of statements).

`restartblock` means that execution restarts at the beginning of the immediately enclosing block.

Example:

```
begin
i:=7;
L1: begin
  j:=j+1;
  if j<i
  L2: then restartblock
  else i:=i+1
  end;
end;
```

`restartblock` at L2 therefore means a jump to L1 (i.e., the beginning of the enclosing block).

a) Write a syntax-directed translation scheme, with attributes and semantic rules, for the above grammar section.

b) What problem would occur in the handling of the translation scheme if instead of `restartblock` there would be an `exitblock` that jumped to the end of the immediate enclosing block (instead of begin).

8. (7p) Intermediate Code Generation and Optimization

Given the following code segment:

```
x = 123;
y = 3;
if (x>100) {
  x = x - y;
  y = 2*y;
}
else
  y = 2*x;
```

```
foo(y);
```

(a) Translate the code segment into abstract syntax trees, quadruples, and postfix code. (4.5 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 (2.5 p)

```
1: T1 := a + b
2: y := T1
3: T2 := - c
4: x := T2 * y
5: T3 := y > 0
6: if T3 goto 13
7: T4 := x < 0
8: if T4 goto 1
10: T5 := x + y
11: y := T5;
12: goto 3
13: m := x * y
```

9. (3 p) Memory management

What does an activation record contain? What happens on the stack at function call? What happens on the stack at function return?

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

(a) Explain the main similarity and the main difference between superscalar and VLIW architectures from a compiler's point of view. Which one is harder to generate code for, and why? (1.5p)

(b) Given the following medium-level intermediate representation of a program fragment:

```
1: c = 3;
2: m = 20;
3: if m<=0 goto 10;
4: a = c / 2;
5: b = a + c;
6: c = a * b;
7: m = m - 1;
8: goto 3;
9: goto 10;
10: d = b * c
```

Identify the live ranges of program variables, and draw the live range interference graph for the entire code fragment.

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!