Försättsblad till skriftlig tentamen vid Linköpings universitet

Datum för tentamen	2016-08-25
Sal (1)	KÅRA
Tid	8-12
Kurskod	TDDD55
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Kompilatorer och interpretatorer En skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	11
Jour/Kursansvarig Ange vem som besöker salen	Jonas Wallgren, endast telefonjur
Telefon under skrivtiden	013-178594
Besöker salen ca klockan	- (endast telefonjour)
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Elin Brödje, 013-28 4767, elin.brodje@liu.se
Tillåtna hjälpmedel	Se tentamens förstasida
Övrigt	
Antal exemplar i påsen	

Tentamen/Exam

TDDB44 Kompilatorkonstruktion / Compiler Construction
TDDD55 Kompilatorer och interpretatorer /
Compilers and Interpreters

2016–08–25, 08.00 – 12.00

Hjälpmedel / Allowed material:

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

General instructions:

- Read all assignments carefully and completely before you begin
- **Note that not every problem is for all courses.** Watch out for comments like "TDDD55 only".
- You may answer in Swedish or in English.
- Write clearly — unreadable text will be ignored. Be precise in your statements — unprecise formulations may lead to 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 6 minutes per point.
- Grading: U, 3, 4, 5 resp. Fx, C, B, A.
- The preliminary threshold for passing (grade 3/C) is 20 points.

1. (TDDD55 only - 6p) **Formal Languages and Automata Theory**
   Consider the language L consisting of all strings w over the alphabet $\{0, 1\}$ such that w contains 11 and does not contain 00. Examples of strings in the language: 01011011010, 111010. Examples of strings not in the language: 01010, 100101.

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

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

   (c) (2.5p) Construct a DFA recognizing L, either by deriving it from the NFA or by constructing it directly.

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

2. (3p) **Compiler Structure and Generators**

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

   (b) (1p) Most modern compilers have not just one, but several intermediate representations(IR). What is the advantage of having more than one IR, and what could be the drawback?

3. (3p) **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 m;
int main( void )
{
  int i;
  // ... some statements omitted
  if (i==0) {
    int j, m;
    // ... some statements omitted
    for (j=0; j<100; j++) {
      int i;
      // ... some statements omitted
      i = m * 2;
    }
  }
}
```

(a) (2p) For the program point containing the assignment i = m * 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 i, value 1 for j and m, and value 4 for main.

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

(c) (0.5p) After code for a block is generated, one needs to get rid of the information for all variables defined in the block. Given a hash table with chaining and block scope control as above, show how to "forget" all variables defined in the current block, without searching through the entire table.

4. (5p) **Top-Down Parsing**
Given a grammar with nonterminals `A`, `B`, and `C`, and `S`, 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) (1p) What is the associativity (right, left, none) of the operators?

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

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

(d) (2p) Write a recursive-descent parser to analyze the language defined by the grammar. (Pseudocode/program code without declarations is fine. Use the function scan() to read the next input token, and the function error() to report errors if needed.)

5. (TDDD55 only - 6p) **LR parsing**

(a) (3p) Use the SLR(1) tables below to show how the string `a+b+b-b` is parsed. You should show, step by step, how stack, input data etc. are changed during the parsing. Start state is `00`, start symbol is `A`.

Grammar:

```
1. A ::= B + B
2.     | B - B
3. B ::= C + C
4.     | C - C
5. C ::= a
6.     | b
```

Tables:

	Action					Goto		
State	+	-	a	b	$	A	B	C
00	*	*	S04	S05	*	01	02	03
01	*	*	*	*	A	*	*	*
02	S06	S07	*	*	*	*	*	*
03	S08	S09	*	*	*	*	*	*
04	R5	R5	*	*	R5	*	*	*
05	R6	R6	*	*	R6	*	*	*
06	*	*	S04	S05	*	*	10	03
07	*	*	S04	S05	*	*	11	03
08	*	*	S04	S05	*	*	*	12
09	*	*	S04	S05	*	*	*	13
10	*	*	*	*	R1	*	*	*
11	*	*	*	*	R2	*	*	*
12	R3	R3	*	*	R3	*	*	*
13	R4	R4	*	*	R4	*	*	*

(b) (3p) Explain the concept of conflict in LR parsing — what it is, how it could be handled.

6. (TDDB44 - 6p) **LR parsing**
Given the following grammar G for strings over the alphabet {a,b,p,q} with nonterminals A, B, and S, where S is the start symbol:

```
A ::= aAa | bAb | aBb | bBa | p
B ::= aBa | bBb | aAb | bAa | q
S ::= A
```

Is the grammar G in SLR(1) or even LR(0)? Justify your answer using the LR item sets.
If it is: construct the characteristic LR-items NFA, the corresponding GOTO graph, the ACTION table and the GOTO table
and show with tables and stack how the string `aabqbba` is parsed.
If it is not: describe where/how the problem occurs.

7. (5p) **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
    x:=17;
L1: begin
        y:=y-42;
        if p=4711
L2:        then restartblock;
           else q:=q-1;
L3: end;
end;
```

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

(a) (4p) Write a syntax-directed translation scheme, with attributes and semantic rules, for translating `<block>`s, and `restartblock`s inside them, to quadruples. The translation scheme should be used during bottom-up parsing. You are not allowed to define and

use symbolic labels, i.e. all jumps should have absolute quadruple addresses as their destinations. You may need to rewrite the grammar. Explain all the attributes, functions, and instructions that you introduce. State all your assumptions. (Since it is a syntax-directed translation scheme, not an attribute grammar, generation of a quadruple puts it in an array of quadruples and attribute values are "small" values such as single quadruple addresses.)

(b) (1p) What problem would occur in handling of the translation scheme if instead of `restartblock` there would be an `exitblock` statement that jumped to the `end` of the immidiately enclosing block (instead of the `begin`), i.e. to `L3` in this example?

8. (3p) **Error Handling**
Explain, define, and give examples of using the following concepts regarding error handling:

   (a) (1p) Panic mode

   (b) (1p) Minimum distance error correction

   (c) (1p) Checked exception

9. (6p) **Intermediate Code Generation**

   (a) (3p) Given the following code segment in a Pascal-like language:

```
if x=y
   then while y>10 do
              if y<x
                 then y:=y+1
                 else y:=y-1
   else x:=func(10);
```

   Translate the code segment into an abtract syntax tree, quadruples, and postfix code.

   (b) (3p) Divide the following code inte basic blocks, draw a control flow graph, and show as well as motivate the existing loop(s).

```
L1: x:=x+1
L2: x:=x+1
L3: x:=x+1
    if x=1 then goto L3
    if x=2 then goto L1
    if x=3 then goto L5
L4: x:=x+1
L5: x:=x+1
    if x=4 then goto L4
```

10. (3p) **Memory management**

    (a) (1p) What does an activation record contain?

    (b) (2p) What happens on the stack at function call and at function return?

    (c) (2p) What are static and dynamic links? How are they used?

11. (TDDB44 only - 6p) **Code Generation for RISC etc.**

    (a) (2p) Explain the main characteristics of CISC and RISC architectures, and their differences.

    (b) (1.5p) 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?

    (c) (1.5p) Explain briefly the concept of software pipelining. Show it with a simple example.

    (d) (1p) What is a live range? Explain the concept and show a simple example.