Försättsblad till skriftlig tentamen vid Linköpings Universitet

Datum för tentamen	2014-01-15
Sal (1) Om tentan går i flera salar ska du bifoga ett försättsblad till varje sal och ringa in vilken sal som avses	KÅRA
Tid	14-18
Kurskod	TDDD55
Provkod	TEN1
Kursnamn/benämning **Provnamn/benämning**	Kompilatorer och interpretatorer En skriftlig tentamen
Institution	IDA
Antal uppgifter som ingår i tentamen	12
Jour/Kursansvarig Ange vem som besöker salen	Jonas Wallgren
Telefon under skrivtiden	– *)
Besöker salen ca kl.	15.00, 17.00
Kursadministratör/kontaktperson (namn + tfnr + mailaddress)	Liselotte Lundberg 281278 liselotte.lundberg@liu.se
Tillåtna hjälpmedel	se tentans första sida
Övrigt	
Vilken typ av papper ska användas, rutigt eller linjerat	rutat
Antal exemplar i påsen	20

*) i tentalokalen, vanligt under tentatiden

Tentamen/Exam

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

2014–01–15, 14.00 – 18.00

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 every occurrence of 00 in w is immediately followed by 11. (Some strings in the language: 11, 10101, 100110, 10100110101001110. Some strings *not* in the language: 00, 10100101.)

   (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) Why is $L_1 = \{a^m b^m c^n d^n\}$ a context-free language but not $L_2 = \{a^m b^n c^m d^n\}$? (For all strings in L_1 the number of a:s and b:s are the same, and the number of c:s and d:s are the same. For all strings in L_2 the number of a:s and c:s are the same, and the number of b:s and d:s are the same.)

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

   (a) (1p) What are the advantages and disadvantages of a multi-pass compiler (compared to an one-pass 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. (2p) **Symbol Table Management**
   Describe what the compiler — using a symbol table implemented as a hash table with chaining and block scoped control — does in compiling a statically scoped, block structured language when it handles:

   (a) (0.5p) block entry

   (b) (0.5p) block exit

   (c) (0.5p) a variable declaration

   (d) (0.5p) a variable use.

4. (5p) **Top-Down Parsing**

   (a) (4.5p) Given a grammar with nonterminals `S`, `A`, and `B`, and the following productions:

   ```
   S ::= 1 S | S 2 | A 2
   A ::= 3 A | 3 B
   B ::= 4 B | 5
   ```

   where `S` is the start symbol and `1`, `2`, `3`, `4`, and `5` are terminals. What is/are the problem(s) with this grammar if it is to be used for writing a recursive descent parser with a single token lookahead? Resolve the problem(s), and write a recursive descent parser for the modified 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.)

   (b) (0.5p) The theory for formal languages and automata says that a stack is required for being able to parse context-free languages. We have used such a stack, for instance, in the LL-item pushdown automaton in the lecture on top-down parsing. But where is the corresponding stack in a recursive descent parser?

5. (6p) **Memory management**

   (a) (1p) What property of programming languages requires the use of activation records?

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

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

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

6. (TDDB44 only - 6p) **LR parsing**
   Given the following grammar G for strings over the alphabet {`x`,`y`,`z`,`t`} with nonterminals `A`, `B`, and `C`, where `A` is the start symbol:

   ```
   A ::= xBx | C
   B ::= ytB | y
   C ::= xBz | y
   ```

   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 `xytyx` is parsed.
   If it is not: describe where/how the problem occurs.

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

(a) (3p) Use the SLR(1) tables below to show how the string `a^b*a+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 S.

Grammar:

```
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
00	*	*	*	*	S11	S10	05	04	08	12
01	*	*	*	*	S11	S10	*	06	08	12
02	*	*	*	*	S11	S10	*	07	08	12
03	*	*	*	*	S11	S10	*	*	*	09
04	*	S01	*	*	*	*	*	*	*	*
05	A	*	*	*	*	*	*	*	*	*
06	R1	*	*	*	*	*	*	*	*	*
07	R2	R2	*	*	*	*	*	*	*	*
08	R3	R3	S02	S03	*	*	*	*	*	*
09	R4	R4	R4	R4	*	*	*	*	*	*
10	R7	R7	R7	R7	*	*	*	*	*	*
11	R6	R6	R6	R6	*	*	*	*	*	*
12	R5	R5	R5	R5	*	*	*	*	*	*

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

8. (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?

9. (2p) **Error Handling**
Explain, define, and give examples of using the following concepts regarding error handling:

(a) (1p) Phrase level recovery,

(b) (1p) Global correction.

10. (6p) **Intermediate Code Generation**

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

```
if fib(x)>fac(y)
   then repeat
          z=z+x;
          x=x+1;
        until z>y or x^2>100;
   else z=0
```

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).

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

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 what register allocation and register assignment is (in the context of code generation), and what the difference is.

(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.

12. (TDDB44 only - 3p) **Compiler Lab Exercises**
Correct and complete labs from the 2012 TDDB44 lab course handed in at the latest December 20, 2013, will give 3 points. State if you think that you have fulfilled the conditions and you should receive these points.