Försättsblad till skriftlig tentamen vid Linköpings universitet

(fylls i av ansvarig)

Datum för tentamen	13-08-26
Sal	TER3
Tid	14-18
Kurskod	TDDB44
Provkod	TEN1
Kursnamn/benämning	Kompilatorkonstruktion
Institution	*IDA*
Antal uppgifter som ingår i tentamen	11
Antal sidor på tentamen (inkl. försättsbladet)	5 blad
Jour/Kursansvarig	Kristian Stavåker
Telefon under skrivtid	076-336 17 82
Besöker salen ca kl.	15.00, 16.30
Kursadministratör (namn + tfnnr + mailadress)	Liselotte Lundberg 28 1278, liselotte.lundberg@liu.se
Tillåtna hjälpmedel	Se tentans förstasida
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	Rutat
Antal exemplar i påsen	12

Tentamen/Exam

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

2013–08–26, 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 w contains 00 or 111 (i.e. at least 2 zeroes in sequence or at least 3 ones in sequence, or both).

   (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) (1p) What are the advantages and disadvantages of a multi-pass compiler (compared to an one-pass compiler)?

   (b) (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?

3. (5p) **Top-Down Parsing**

   (a) (4.5p) Given a grammar with nonterminals `<Expr>`, `<SimpExpr>`, `<ArrayRef>`, and `<IndexExprs>` and the following productions:

   ```
   <Expr> ::= Id | <ArrayRef>
   <SimpExpr> ::= Id | Colon
   <ArrayRef> ::= Id[<IndexExprs>]
   <IndexExprs> ::= <IndexExprs>,<SimpExpr> | <SimpExpr>
   ```

   where `<Expr>` is the start symbol, the comma character(`,`), `Id` and `Colon` are terminals. (This may e.g. generate expressions such as `arr[c,:,:,d,:]` were `arr`, `c`, and `d` are identifiers—the colons could be specifying slices.)

   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?

4. (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.

5. (3p) **Error Handling**
Explain, define, and give examples of using the following concepts regarding error handling:

(a) (1p) Valid prefix property,

(b) (1p) Phrase level recovery,

(c) (1p) Global correction.

6. (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	*	*	*	*	S09	S10	01	02	05	08
01	A	*	*	*	*	*	*	*	*	*
02	*	S03	*	*	*	*	*	*	*	*
03	*	*	*	*	S09	S10	*	04	05	08
04	R1	*	*	*	*	*	*	*	*	*
05	R3	R3	S06	S11	*	*	*	*	*	*
06	*	*	*	*	S09	S10	*	07	05	08
07	R2	R2	*	*	*	*	*	*	*	*
08	R5	R5	R5	R5	*	*	*	*	*	*
09	R6	R6	R6	R6	*	*	*	*	*	*
10	R7	R7	R7	R7	*	*	*	*	*	*
11	*	*	*	*	S09	S10	*	*	*	12
12	R4	R4	R4	R4	*	*	*	*	*	*

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

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

```
X ::= aX | Xb | aYb | p
Y ::= bY | Ya | bXa | q
```

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

8. (5p) **Syntax-Directed Translation**
An Algol-like language is augmented with an if2-statement in the following way:

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

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

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

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.

9. (3p) **Memory management**

   (a) (1p) Non-local references: How does a static link work?

   (b) (1p) Non-local references: How does a display work?

   (c) (1p) Dynamic data: How is the actual size and contents of a dynamic array handled?

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

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

```
if x<y and y<z
   then repeat
          y:=y+k
        until x<y or y<z
   else print(func(y));
```

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

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

   (a) (1p) 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?

   (b) (2p) What is branch prediction and when is it used? Give an example! Why is this important for pipelined processors?

   (c) (3p) Given the following medium-level intermediate representation of a program fragment:

```
1: a = 1.0
2: b = 1.0
3: c = 3.0
4: e = 2.0
5: goto 9
6: b = a + b
7: a = c / 2.0
8: c = a * e
9: e = e / 2.0
10: f = (e > 0.1)
11: if f goto 6
12: d = 1.5 * a
```

Identify the live ranges of program variables, and draw the live range interference graph for the entire fragment. Assign registers to all live ranges by coloring the live range interference graph. How many registers do you need at least, and why?