Norwegian University of Science and Technology
NTNU

Department of Computer and Information Science

FINAL EXAM [Eksamen]
TDT 4205 Compiler Technology [Kompilatorteknikk]
Monday, May 30th , 2011
Time: 15:00-19:00

Instructional contact during the exam

[faglig/fagleg kontakt under eksamen]:

Jan Christian Meyer (mob. 41 19 32 64 kontor 73 59 05 42)

Aids [hjelpemidler]: **C**
Specific printed and handwritten aids

[Spesifikke trykte og håndskrevne hjelpemidler/

Særskilde trykte og handskrivne hjelpemiddel]

- **1 (one) stamped A4-size sheet of handwritten notes on both sides**
  [1 (ett) stemplet A4-ark med håndskrevne notater på begge sider/
  1 (eitt) stempla A4-ark med handskrivne notatar på begge sider]

- **Assembler syntax summary, included in the exam set**
  [Sammendrag av assemblersyntaks, innskrevet i eksamenssettet]

Grades will be assigned within three working weeks.
[Karakterer vil bli satt innen tre arbeidsuker /
Karakterar vil bli satt innan tre arbeidsveker]

It is NOT necessary to justify your answer on true/false questions.
[Det er ikke nødvendig å begrunne TRUE/FALSE spørsmål. /
Det er ikkje naudsynt å grunngje TRUE/FALSE spørsmål.]

(If english and norwegian translations differ, english should take precedence.)
([Om det finnes/finst forskjell på engelsk/norsk oversetting, er den engelske foretrukket])

CANDIDATE NUMBER: 77

(Suggested solutions)

1 True / False [Sant/usant] (10 / 80)

Incorrect answers will receive a negative score. [Ukorrekte svar gir negative poeng.]

a) With a strongly typed language, type checking will recognize all type-safe programs

[Med ett/eitt sterkt typa språk vil typesjekking gjenkjenne alle typesikre program]

TRUE/FALSE

b) A given language corresponds to a unique context-free grammar

[Et/eit gitt språk svarer til en/ein unik kontekstfri grammatikk] TRUE/FALSE

c) LALR(k) parsing can give smaller tables than LR(k)

[LALR(k)-parsing kan gi/gje mindre tabeller/tabellar enn LR(k)] TRUE/FALSE

d) The meet-over-paths solution to a set of dataflow equations is always at least as precise as the maximal fixed point solution

[Meet-over-paths-losningen/loysinga til et/eitt sett med dataflytligninger/likningar er alltid minst like presis som maximal fixed point-losningen/loysinga] TRUE/FALSE

e) Variables which are connected in an interference graph can share a register

[Variabler/variablar som er sammenbundet/samanbundne i en/ein interferensgraf ~~kan~~ dele et/eit register] TRUE/FALSE

Not in syllabus 2016

f) Every deterministic finite automaton is also a nondeterministic finite automaton

[Hver/kvar deterministiske tilstandsmaskin er også en/ein ikkedeterministisk tilstandsmaskin]

TRUE/FALSE

g) Several different lexemes can correspond to the same token

[Flere/fleire ulike leksemer/leksem kan svare til samme/same token] TRUE/FALSE

h) Moving loop-invariant code to a loop preheader alters the semantics of the program

[Flytting av lokkeinvariant kode til preheader endrer semantikken i programmet]

TRUE/FALSE

i) Grammars with left-recursive productions are not LL(k) parseable for any value of k

[Grammatikker/grammatikkar med venstrerekursive produksjoner/produksjonar er ikke/ikkje LL(k)-parselige for noen/nokon k] TRUE/FALSE

j) At the last control flow point before a return statement, no more than 1 variable may be live

[Ved siste kontrollflytpunkt for return-punkt kan ikke/ikkje mer/meir enn 1 variabel ~~være~~/vere live]

TRUE/FALSE

2. Grammars [Grammatikk] (15/80)

a) Rewrite this grammar for LL(1) parsing, by left factoring it and eliminating left recursion.
[Skriv om denne grammatikken for LL(1)-parsing, ved venstrefaktorering og eliminering av venstrerekursjon]

S → s C T | s C T w B
C → c
T → t | ϵ
B → B a | a

b) Tabulate the FIRST and FOLLOW sets of the nonterminals in the resulting grammar, and construct the predictive parsing table.
[Tabuler FIRST og FOLLOW for nonterminalene/nonterminalane i resultatet, og konstruer prediktiv parsetabell]

S → s C T S'
S' → w B | ϵ
T → t | ϵ
B → a B'
B' → a B' | ϵ

	S	S'	T	B	B'
FIRST	s	w	t	a	a
FOLLOW	$	$	$, w	$	$
nullable	no	yes	yes	no	yes

	s	w	t	a	$
S	S → s C T S'				
S'		S' → w B			S' → ϵ
T		T → ϵ	T → t		T → ϵ
B				B → a B'	
B'				B → a B'	B → ϵ

c) By default, yacc resolves shift/reduce conflicts by shifting. Briefly explain the consequence of resolving these by reducing instead, using the dangling-else ambiguity as an example.
[Yacc løser shift/reduce-konflikt ved shift. Forklar kort hvilke følger det vil ha å redusere isteden/istaden, og bruk dangling-else-ambiguity som eksempel.]

The dangling else ambiguity admits two interpretations of a nested if-statement

(1) if(a) [if(b) {x} else {y}] (else paired w. inner if(b))

(2) if(a) [if(b) {x}] else {y} (else paired w. outer if(a))

Resolving this ambiguity by shifting selects (1), whereas reducing selects (2).

Files in the NIST matrix exchange format begin with a header of 3 words from 3 sets, which are (in order) {coordinate, array}, {real, integer, complex, pattern}, and {general, symmetric, skew-symmetric, Hermitian}. If the last word is 'Hermitian', the 2nd word must be 'complex'. If the 2nd word is 'pattern', the 1st word must be 'coordinate', and the 3rd must be either 'general' or 'symmetric'.

[Filer i NIST matrix exchange-formatet starter med en innledning ("header") på 3 ord fra 3 mengder, i ordnet rekkefølge {coordinate, array}, {real, integer, complex, pattern}, og {general, symmetric, skew-symmetric, Hermitian}. Dersom det siste ordet er 'Hermitian', må det andre ordet være 'complex'. Dersom det andre ordet er 'pattern', må det første ordet være 'coordinate', og det tredje enten 'general' eller 'symmetric'.]

d) Write a context-free grammar for the specified header format.

[Skriv en kontekstfri grammatikk for det spesifiserte headerformatet.]

$H \rightarrow G_1\ G_2\ G_3 \mid G_1$ complex $G_5 \mid$ coordinate pattern G_3

$G_1 \rightarrow$ coordinate | array
$G_2 \rightarrow$ real | integer
$G_3 \rightarrow$ general | symmetric
$G_4 \rightarrow G_3 \mid$ skew-symmetric
$G_5 \rightarrow G_4 \mid$ Hermitian

e) Can this header be recognized by a regular expression? Why/why not?
[Kan slike headere gjenkjennes av et regulært uttrykk? Hvorfor/hvorfor ikke?]

Yes, it can. The language contains at most a finite number of combinations made from fixed, finite words, so a regular expression can simply contain all combinations that are valid, separated by selection ("or"-operators).

3. Data flow analysis [Dataflytanalyse] (15/80)

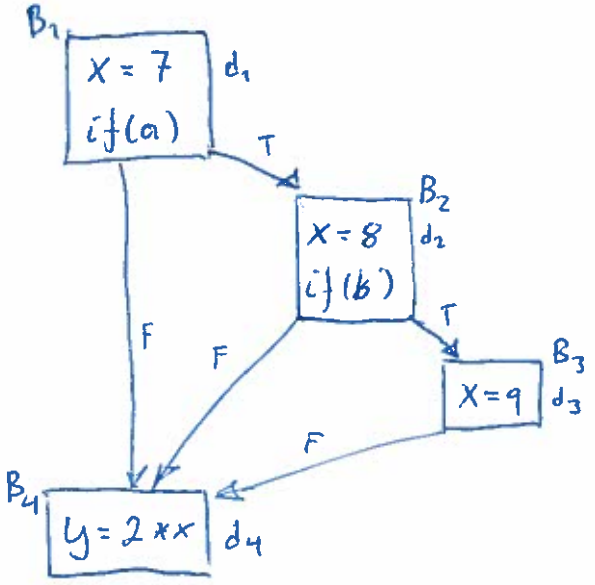
Consider the following program fragment
[Se/sjå på folgende/folgjande programfragment]:

```
x = 7
if ( a ) {
    x = 8
    if ( b ) {
        x = 9
    }
}
y = 2 * x
```

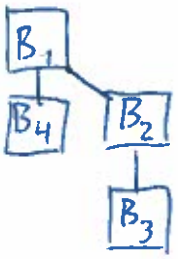
a) Draw its control flow graph
[Tegn/teikn kontrollflytgrafen]

b) Label the blocks, and draw the dominator tree
[Merk hver/kvar blokk, og tegn/teikn dominatortreet]

c) Number the appropriate statements, and label the graph with corresponding *in*, *out*, *gen* and *kill* sets for reaching definitions analysis. (Block level is sufficient, it is not necessary to show the control points before and after every statement.)
[Nummerer riktige utsagn og merk grafen med samsvarende in, out, gen og kill-mengder for Reaching Definitions. (Blokknivå er tilstrekkelig/tilstrekkeleg, det er ikke nodvendig / naudsynt å vise kontrollpunkt for og etter hvert/kvart utsagn)]



Dominator tree



	in	out	gen	kill
B_1		d_1	d_1	*
B_2	d_1	d_2	d_2	d_1
B_3	d_2	d_3	d_3	d_2
B_4	d_1, d_2, d_3	d_1, d_2, d_3, d_4	d_4	**

* any defs. of x, but none are visible

** any defs. of y, but none are visible

(Not necessary to state)

d) What is the significance of having a *monotonic* transfer function?
[Hvilken/kva innflytelse har det at transferfunksjonen er monoton?]

Monotonicity guarantees that iterative application of the transfer function will reach the maximal fixed point for each program point (provided that the analysis initializes these with a representation of the topmost point in a lattice order).

e) What is the significance of having a *distributive* transfer function?
[Hvilken/kva innflytelse har det at transferfunksjonen er distributiv?]

A distributive transfer function guarantees that the maximal fixed point solution (obtained iteratively) equals the meet-over-paths solution (which is intractable to compute in the general case).

4. Miscellaneous [Diverse] (15/80)

a) In C, *sqrt* is an external library function, whereas in FORTRAN, it is an intrinsic operation defined by the language. Briefly explain which difference this makes to an optimizing compiler when analyzing a loop like the following one:
[I C er sqrt et/eit eksternt bibliotekskall, i FORTRAN er det en/ein innebygd operasjon I språket. Forklar kort hvilken/kva forskjell dette utgjor for kompilatoroptimalisering ved analyse av ei lokke som denne]:

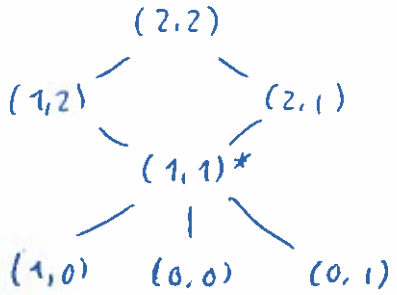
```
x = 2.0
for ( i=0; i<N; i++ )
   a[i] = b[i] * sqrt(x)
```

With the def. of sqrt as part of the language, the value of sqrt(x) may be recognized as loop-invariant, and moved out of the iteration / substituted with its constant value.

With an external library function this can not be safely done unless some additional guarantee is made that the function does not retain internal state, causes side effects, or manipulates global state.

b) Define a relation for ordered pairs of integers { (a,b), (c,d) }, such that (a*b) < (c*d). Draw a Hasse diagram for the order this imposes on the set of pairs { (0,0), (1,0), (0,1), (1,1), (1,2), (2,1), (2,2) }.

*[Definer en/ein relasjon for ordna heltallspar {(a,b),(c,d)}, slik at (a*b) < (c*d). Tegn/teikn et/eit Hasse-diagram for ordenen dette pålegger parmengden { (0,0), (1,0), (0,1), (1,1), (1,2), (2,1), (2,2) }.]*



*glb{(1,2),(2,1)}

c) Mark the greatest lower bound for (1,2) and (2,1).
[Marker storste nedre beskrankning for (1,2) og (2,1)]

d) Does this pair of set and ordering relation form a lattice? Justify your answer.
[Danner dette mengde/orden-paret et gitter? Begrunn svaret.]

This is not a lattice, there is no greatest lower bound for tuples of pairs taken from {(1,0), (0,0), (0,1)}

e) Given the inference rules *[Gitt slutningsreglene/reglane]*

E1: T E2: T / E1+E2: T

E1:T E2:T / E1>E2: bool

C:bool E1:T E2:T / (C) ? E1:E2

and the premises that **2:int** and **3.14:float**, show a proof tree with judgements on the types of x and y in the statement

*[og premiss **2:int** og **3.14:float**, vis et/eit bevistre med judgement på typene/typane til x og y i utsagnet]*

(x>2) ? y : 3.14

$$\dfrac{\dfrac{\dfrac{}{x:int}\quad\dfrac{}{2:int}}{x:T_1 \quad 2:T_1}}{(x>2):bool}\qquad\dfrac{\dfrac{}{y:float}\quad\dfrac{}{3.14:float}}{y:T_2 \quad 3.14:T_2}$$

$$(x>2)\ ?\ y : 3.14$$

5. Programming [Programmering] (25/80)

The scanner/parser pair on the following pages specify a small language from strings of operations MOVE, LOAD, SET, GET, ADD, SUB, MUL, DIV, DO, WHILE, OUT and HALT. The language works on an array of 32-bit integers, using a value register and a position. The state of the machine includes a pointer to the array element it is currently working on, and the value presently contained in the register. Operations combine the register value and the current array element.

[Scanner/parser-paret på de neste sidene spesifiserer et lite språk definert av strenger av operasjonene MOVE, LOAD, SET, GET, ADD, SUB, MUL, DIV, DO, WHILE, OUT og HALT. Språket arbeider med et array av 32-bits heltall, ved bruk av et verdiregister og en posisjon. Maskinens tilstand inkluderer en peker til array-elementet den arbeider på for oyeblikket, og verdien i registeret. Operasjoner kombinerer registerverdien og det gjeldende arrayelementet.]

Instruction semantics *[Instruksjonenes semantikk]*

MOVE <integer>:	Shift the position in the array by <integer> 32bit values *[Skift arrayposisjon med <integer> 32bit-verdier]*
LOAD <integer>:	Set the value of the register to <integer> *[Sett registerverdien til <integer>]*
SET / GET:	Set the array element to the register value, and vice versa *[Sett arrayelementet til registerverdien, og vice versa]*
ADD / SUB / MUL / DIV:	Modify the array element by the register value using the corresponding arithmetic operation, storing the result in the current array element *[Endre arrayelementet med registerverdien ihht. tilsvarende aritmetisk operasjon, og lagre resultatet i arrayelementet]*
DO / WHILE:	When reaching WHILE, control returns to the corresponding DO if the value in the current array element is different from zero. *[Når kontrollen når WHILE, returnerer den til den tilsvarende DO dersom verdien i arrayelementet er ulik null]*
OUT:	Print the contents of the current array element on std. output *[Skriv ut innholdet i arrayelementet på std. Output]*
HALT:	End the program *[Avslutt programmet]*

Your task is to complete the parser by filling in the blank semantic actions on pages 13,14, so that it translates programs in this language into IA-32 assembly.

[Oppgaven er å fullføre parseren ved å fylle inn de blanke semantiske handlingene på sidene 13,14, slik at den oversetter programmer i dette språket til IA-32 assembly.]

Notes:

- The pointer to the first array element is already initialized in the EBX register.
- The INTVAL macro gets the integer value of the text in the the scanner's yytext buffer.
- The DO production is already implemented: it declares a label, and pushes the address of that label on stack. These addresses can be used as jump targets, e.g. the syntax "`jmp *%eax`" jumps to the address contained in EAX. Note that this mechanism is restricted to unconditional jumps.
- The scanner requires no modification, but is included for the sake completeness.
- The program examples on page 11 are included to illustrate the operation of the language, by stating a simple program of double-nested counter loops, and an equivalent C program.

[Merknader:

- *Pekeren til første arrayelement blir allerede initialisert i EBX-registeret*
- *Makroen INTVAL henter heltallsverdien fra teksten i scannerens yytext-buffer*
- *DO-produksjonen er allerede implementert: den erklærer en label, og skyver addressen til denne på stack. Disse addressene kan brukes i hoppinstruksjoner, f.eks. vil syntaksen "`jmp *%eax`" hoppe til addressen som er lagret i EAX-registeret. Merk at denne mekanismen er begrenset til ubetingede hopp*
- *Scanneren behover ingen endring, men er vedlagt for fullstendighetens skyld*
- *Programeksemplene på s. 11 er inkludert for å illustrere hvordan språket opererer, ved å vise et enkelt program av dobbelt nostede tellerløkker, og et ekvivalent C-program*

]

Examples *[Eksempler]* :

```
MOVE 1                  // Move one cell to the right
LOAD 50 SET             // Set 50 in cell 1
DO                      // Outer loop, from 50 to 0 at stride -10
    OUT                 // Print the counter for the outer loop
    MOVE -1             // Move to cell 0
    LOAD 3 SET          // Set 3 in cell 0
    DO                      // Inner loop, from 3 to 0
        OUT                 // Print the inner loop counter
        LOAD 1 SUB          // Subtract one from the counter (cell 0)
    WHILE                   // Loop while cell 0 is != 0
    MOVE 1              // Move to cell 1
    LOAD 10 SUB         // Subtract 10
WHILE                   // Loop while cell 1 is != 0
HALT                    // Stop.
```

```
#include <stdio.h>
#include <stdlib.h>

int a[2], c = 0;

int main ( int argc, char **argv )
{
    c += 1;
    a[c] = 50;
    do
    {
        printf ( "%d\n", a[c] );
        c -= 1;
        a[c] = 3;
        do
        {
            printf ( "%d\n", a[c] );
            a[c] = a[c] - 1;
        } while ( a[c] );
        c += 1;
        a[c] = a[c] - 10;
    } while ( a[c] );
    exit ( EXIT_SUCCESS );
}
```

Small IA32 instruction reminder:

[Liten påminnelse om noen IA32-instrukser:]

`movl <src>, <dst>` - move src value to dst [Flytt src til dst]

`addl <src>,<dst>` - add src value to dst [Legg verdien i src til dst]

`imull <src>` - multply 64-bit value in %edx:%eax by src
[Gang 64-bitsverdien %edx:%eax med src]

`idivl <src>` - divide %edx:%eax by src, store quotient in %eax, remainder in %edx
[Divider %edx:%eax med src, plasser kvotienten i %eax, restleddet i %edx]

`cdq` - sign extend %eax to %edx:%eax [Utvid fortegnet i %eax til %edx:%eax]

`pushl <src>` - push src on stack [Skyv src på stakk]

`popl <dst>` - pop value from stack to dst [Hent øverste/øvste verdi på stakk til dst]

Some registers and their roles: [Noen registre og rollene deres:]

`%eax` - results accumulator [Resultatakkumuluator]

`%ebx` - general data register [Generelt dataregister]

`%esp` - stack pointer [Stakkpeker / stakkpeikar]

`%ebp` - frame pointer [Rammepeker / rammepeikar]

Addressing modes: [Adresseringsmodi:]

`%eax` - register EAX [register EAX]

`(%eax)` - memory contents at addr. EAX [minnets innhold ved adr. EAX]

Parser.y:

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
extern char *yytext;
static int labelcount = 0;

#define HEAD                    \
".data\n"                       \
".ARRAY: .fill 256,4,0\n"       \
".OUT: .string \"%%d\\n\"\n"\
".globl main\n"                 \
".text\n"                       \
"main:\n"                       \
" pushl %%ebp\n"                \
" movl %%esp,%%ebp\n"           \
" movl $.ARRAY,%%ebx\n"

#define TAIL                \
" leave\n"                  \
" movl $0,%%eax\n"          \
" ret\n"

#define INTVAL ((int32_t)strtol(yytext,NULL,10))
%}
%token INTEGER MOVE LOAD SET GET ADD SUB MUL DIV DO WHILE OUT HALT
%%
program: statement_list HALT {}
statement_list: statement | statement statement_list
statement:
    MOVE INTEGER { printf("addl $%d, %%ebx\n", 4*INTVAL); }
  | LOAD INTEGER { printf("movl $%d, %%eax\n", INTVAL); }
  | SET { printf("movl %%eax, (%%ebx)\n"); }
  | GET { printf("movl (%%ebx), %eax\n"); }
  | ADD { printf("addl %%eax, (%%ebx)\n"); }
  | SUB { printf("subl %%eax, (%%ebx)\n"); }
```

```
| MUL  { printf("xchgl %%eax,(%%ebx)\n"
             "cdq\n"
             "imull (%%ebx)\n"
             "xchgl %%eax,(%%ebx)\n"
       );
     }
| DIV
     {   printf ("xchgl %%eax,(%%ebx)\n"
             "cdq\n"
             "idivl (%%ebx)\n"
             "xchgl %%eax,(%%ebx)\n"
     );
     }
| DO  {
      printf ( "do%d:\npushl $do%d\n", labelcount, labelcount );
      labelcount += 1;
     }
| WHILE {
     printf ( "cmpl $0,(%%ebx)\n" );
     printf("je break%d\n", labelcount);
     printf("popl %%ecx\n"
            "jmp *%%ecx\n"
     );
     printf("break%d:\n", labelcount );
     printf("addl $4,%%esp\n");
     labelcount += 1;
     }
| OUT  {
     printf("pushl (%%ebx)\n"
            "pushl $.OUT\n"
            "call printf\n"
            "addl $8,%%esp\n"
     );
     }
     ;
%%
```

```
int yyerror(void) { puts ( "Syntax error" ); exit ( EXIT_FAILURE ); }
int
main ( int argc, char **argv )
{
    printf ( HEAD );
    yyparse();
    printf ( TAIL );
    exit ( EXIT_SUCCESS );
}
```

Scanner.l

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "y.tab.h"
%}
%option noyywrap
%%
\/\/[^\n]+\n { /* Line comments */ }
[\ \t\n]    { /* Whitespace */ }
-?[0-9]+   { return INTEGER; }
MOVE     { return MOVE; }
LOAD     { return LOAD;  }
SET      { return SET;   }
GET      { return GET;   }
ADD      { return ADD;   }
SUB      { return SUB;   }
MUL      { return MUL;   }
DIV      { return DIV;   }
DO       { return DO;    }
WHILE    { return WHILE;}
OUT      { return OUT;   }
HALT     { return HALT; }
.        { return yytext[0]; }
%%
```