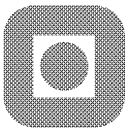
STUDENT NUMBER: _____

Norwegian University of Science and Technology
NTNU

Department of Computer and Information Science



FINAL EXAM [Eksamen]
TDT 4205 Compiler Technology [Kompilatorteknikk]
Wednesday, May 30, 2012
Time: 09:00-13:00

Instructional contacts [faglige kontakter] during the exam:

Anne C. Elster (918 97 062)

Aids [hjelpemidler]: C (ingen hjelpemidler) bortsett fra ett håndskrevet notatark som er stemplet med "NTNU og Department of Computer and Information Science" som skal innleveres med alle svarene som skal være innskrevet på oppgavesettet. Assembler syntaksen du trenger er i år skrevet inn på disse eksamensarkene.

No aids except one NTNU stamped sheet of handwritten notes that need to be returned with this exam set along with all answers which should be given in the space provided on this exam set. This year the essential assembler syntax you need is integrated with this exam set.

[Ikkje nokre hjelpemidlar bortsett frå eit ark med handskrevne notatar som skal være stempla med "NTNU og Department of Computer & and Information Science". Dette arket skal innleverast saman med dette eksamenssettet kor du og skal gje alle svara der det er laga plass til det. I år har eg skreve assemblersyntaksen inn i disse eksamensarka for dere.]

Grades will be assigned within three working weeks.

[Karakterer vil bli satt innen tre arbeidsuker]
[Karekterar vert sette før tre airbeidsveker er omme]

It is NOT necessary to justify your answer on true/false questions.
[Ein treng ikkje grunngje/En trenger ikke å grunngi TRUE/FALSE spørsmål.]

GIVE ALL ANSWERS ON THE FOLLOWING PAGES
(Før alle svara inn på dette oppgave/oppgåve-settet!

If there seems to be inconsistencies between the English and the Norwegian quesgion, the English one should be considered (engelsk er førenede).

1. Warm-ups [oppvarming] – TRUE/ FALSE [Sant/u sant] (20%)

Circle your answers -- Note: You will get a negative score for wrong answers and 0 for not answering or circling both TRUE and FALSE.

[Sett sirkel rundt svara -- NB: Dere får negative poeng om dere svarer feil, 0 poeng for å ikke/ikkje svare eller sirkle både "TRUE"(sant) og "FALSE" (usant).]

a) Code generation are part of the front-end TRUE/FALSE
[kodegenerering er del av en/ein kompilator kalles/er og kjend som "front-end"]

b) Bison produces tokens ["Bison" lager "tokens"] TRUE/FALSE

c) Associativity rules will always resolve operator ambiguity TRUE/FALSE
[Assosisativitetsregler vil alltid ta hand om operator tvetydlighet/ning]

d) Semantic analyzers use the symbol table and do type checking TRUE/FALSE
[semantiske analysatorer bruker symboltabeller og utfører typesjekk]

e) Top-down parsers do not need to figure out as much of the parse tree or a given amount of input compared to bottom-up parsers TRUE/FALSE

f) Heap data doesn´t explicitly occur in low-level code TRUE/FALSE
["heap"/haug data finnast ikke eksplisit i lav/låg-nivå kode]

g) Both Java and C allow type aliases
[Både Java og C godtar "type aliases"] TRUE/FALSE

h) new(Java/C++) or malloc (C) allocate variable space directly on stack TRUE/FALSE
[new(Java/C++) og malloc(C) alllokerer variabelplass direkte på stakken]

i) A frame (activation record) includes local variables TRUE/FALSE
[En "activation record" inkluderer lokale variabler]

j) Derived induction variables are only incremented in loop body TRUE/FALSE
[utledete induksjonvariable er kun inkrementert innen løkkene]

1. CONTINUED (Fortsettelse av Oppg. 1):

k) Type conversion may not be done automatically by the compiler TRUE/FALSE
[type-konverteringer kan ikke gjøres/gjerast av kompilatoren]

l) Static typing implies strong typing TRUE/FALSE
[Statisk typing fører til sterk typing]

m) Dataflow analysis is not part of code optimization TRUE/FALSE
[Dataflytanalyse er ikke/ikkje en/ein del av kodeoptimisering]

n) Tiling w.r.t. instruction selection involves matching tree pattern to corresponding machine instruction TRUE/FALSE
["Tiling" m.h.t. valg av instruksjoner handler om å matche tremøstre til tilsvarende maskinkode]

o) Moving loop-invariant code to a loop preheader alters the semantics of the program TRUE/FALSE
[Flytting av løkkeinvariant kode til preheader endrer semantikken i programmet]

p) Dominators in CFGs are used to identify loops TRUE/FALSE
["Dominators" i kontrollflytgrafer er nyttet til å identifisere løkker]

q) Tree representation may be appropriate for instruction selection TRUE/FALSE
[Tree representasjoner kan være rett for instruksjonsutvalg.]

r) The Maximal Munch algorithm finds the least expensive tiling of a DAG ["Maksimal Munch" algoritmen finner den minst dyre "tiling" av DAG]
TRUE/FALSE

s) The x86 assembler takes 3-address code as input.
[x86-assembleren tar 3-addressekode som input*]* TRUE / FALSE

t) In short-circuit code boolean operators translate into jumps TRUE/FALSE
[I "short-circuit"/kortslutnings-koder oversettes boolske operatorer til hopp]

2. COMPILER BASICS. (20%) Fill in the blanks (fyll inn svarene)

a) **What does a Lexer do? [Hva gjør/gjer en lexer?]**

_____

a) What does Bison do? [Hva gjør/gjer Bison?]

_____

b) **What kind of information is stored in a syntax trees?**
[Hva slags /Kva for type info. er lagra i et/eit syntaxtre?]

_____

c) What is a syntax-directed definition (SDD)? [Hva/Kva er SDD?]

_____

d) **What is the main purpose of Dataflow analysis?**
[Hva er hoved/hovud-poenget med dataflyt-analyse?]

_____

e) **How is tiling used with respect to instruction selection?**
[Hvordan/korleis er ”tiling” brukt/bruke vis á vis instruksjonsselektering?]

_____

f) **State two advantages obtained by function inlining.**
[Oppgi to fordeler som oppnås ved inlining av funksjoner. / Oppgje to fordelar som ein oppnår ved inlining av funksjonar.]

_____

_____

g) **What is backpatching?** [Hva/Kva er “backpatching”?]

_____

h) **What does the The Maximal Munch algorithm do?**
[Hva/Kva gjør/gjer Maximal Much algoritma?]

_____

i) **Describe an example where loop unrolling would decrease performance:**
[Beskriv/Gje et/eit eksempel der utrulling av løkken vil svekke ytelsen]

3. OPTIMIZATION [Optimering] (10%)

3 a)

a=1
b=2
c =3
d=a+x
e=b+c
f =e
g=f
g=d+y
a=b+c

In each of the cases below, state the optimization(s) applied to the code above in optimizing the above basic block.
[For hver/kvar av tilfellene/høvene nedenfor, angi optimeringen(e) av basisblokka ovenfor

i)
a=1
b=2
c =3
d=a+x
e=b+c
f =e
g=e
g=d+y
a=b+c

ii)
a=1
b=2
c =3
d=1+x
e =5
f =5
g=5
g=d+y
a=5

iii)
a=1
b=2
c =3
d=a+x
e=b+c
t 1=e
f =e
g=f
g=d+y
a=t 1

iv)
a=1
b=2
c =3
d=a+x
e=b+c
f =e
g=d+y
a=b+c

i) _____

ii)_____

iii) _____

iv) _____

3b) Dominator tree

Draw the corresponding Dominator Tree next to the given flow graph.

[Tegn/tekn korresponderenede dominator treet ved siden av flytgrafen nedenfor)

(from Pg 657)

4. Problem Set Related Questions [Øvingsrelaterte Oppg.] (30%)

a) Related to PS1: From Kernighan & Ritchie, p194:
[Fra/frå Kernighan & Ritchie, s. 194:]
tte formatet]

"A floating constant consists of an interger part, a decimal point, a fraction part, an "e" or "E", an optionally signed integer exponent and an optional type suffix, one of "f", "F", "l", or "L". The integer and fraction parts both consists of a sequence of digits. Either the integer part or the fraction part (not both) may be missing: Either the decimal point or the "e" and the exponent (not both) may be missing. The type is determined by the suffix: "F" or "f" makes it float, "l" or "L" makes it long double: Otherwise it is double."

Draw a deterministic finite automaton which accepts floating point constants in this format

[Tegn et/eit deterministisk "finite automaton" so aksepterer/ar "floating point constants" i dette formatet]

4 b) From PS 2 (and related topics)

i) What makes a bottom-up parser more powerful than a top-down parser?

ii) Is there a regular language which can be recognized by an NFA but not by any DFA?
Explain why/why not.

iii) What kind of parser does GNU Bison generate?

4 c) From PS 3: Parsing by recursive descent

Consider the grammar :

P → B
B → ET
E → DE | ε
T → TS | ε
D → tiV
V → v | ε
S → k | bBc

i) Tabulate FIRST and FOLLOW for each non-terminal

.

ii) **What kind of data structure is typically used for symbol tables, and why**

d) From PS 5:

i**) What is the difference between forward and backwards analysis with respect to the topic of compiler optimization?**

ii) Task 1.1 - Data flow analysis

The code for bubble sort is given as

```
for(int x=0; x<n; x++) {

      for(int y=0; y<n-1; y++) {
            if(array[y]>array[y+1]) {
                  int temp = array[y+1];
                  array[y+1] = array[y];
                  array[y] = temp;
            ]
      ]
}
```

Assume that the variables array and n has been defined earlier, and indices are valid from 0 to n-

4.ii.2. Circle the optimizations you can perform on the expected 3-address code for the C code above.

:

• Global and local common subexpression elimination

• Copy propagation

• Constant folding (if any)

• Dead code elimination

4.ii.3 Re-write the inner loop in IA32 assembler (see below for instruction reminder)

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]
cltd	- 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 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]

%ebp - frame pointer [Rammepeker]

Inner loop:

```
int temp = array[y+1];
                array[y+1] = array[y];
                array[y] = temp;
```

Space fore assembly code:

5 FOR LOOPS (20%)

This problem entails implement for-loops fort VSL. An example of us is as follows:

[Denne oppgaven/oppgåva går ut på å implementere/a for-løkker for VSL. Et /Eir eksempel på bruk av for-løkker er:]

```
VAR i, j

FOR i := 1 TO 10 DO
   FOR j := i+1 TO 2*i DO
     PRINT "i is ", i, ", j is ", j
     PRINT "And that's the way the cookie crumbles."
.  DONE
DONE
```

5a) Grammar

Suggest a grammar rule for the for-statement, given the VSL grammar from the problem sets (and included on the next page).

[Foreslå en/ein grammatovekkregel for for-statement, gitt VSL-grammatikken fra/frå øvingene/-ane (og lagt med på neste side]:

From[fra/frå PS 2:

program → function_list
function_list → function | function_list function
statement_list → statement | statement_list statement
print_list → print_item | print_list ‘, ’ print_item
expression_list → expression | expression_list ‘, ’ expression
variable_list → variable | indexed_variable | variable_list ‘, ’ variable |
variable_list ‘, ’ indexed_variable
argument_list → expression_list | ε
parameter_list → variable_list | ε
declaration_list → declaration_list declaration | ε
function → FUNC variable ‘(’ parameter_list ‘)’ statement

statement → assignment_statement | return_statement | print_statement | null_statement |
if_statement | while_statement | block

block → ‘{’ declaration_list statement_list ‘}’
assignment_statement → variable ASSIGN expression |
variable ‘[’ expression ‘]’ ASSIGN expression
return_statement → RETURN expression
print_statement → PRINT print_list
null_statement → CONTINUE

if statement → IF expression THEN statement FI |
IF expression THEN statement ELSE statement FI

While_statement → WHILE expression DO statement DONE

expression → expression ‘+’ expression | expression ‘-’ expression | expression ‘*’ expression |
expression ‘/’ expression | ‘-’ expression | expression POWER expression |
‘(’ expression ‘)’ | integer | variable | variable ‘(’ argument_list ‘)’ | variable ‘[’ expression ‘]’

declaration → VAR variable_list
variable → IDENTIFIER
indexed_variable → variable ‘[’ integer ‘]’
integer → NUMBER
print_item → expression | text text → STRING

Figure 1 from PS 2: Context Free Grammar of VSL
3

5b) Implementation

Implement the code generation for the `for` statement above. Assume that the rest of the VSL language is already implemented. Include the code below and/or on the following sheet.

[Implementer kodegenerering for for-statement.
Anta at resten av VSL-språket allerede er implementert.
Skriv ned koden nedenfor og/eller på neste side]