Örebro universitet
Institutionen för naturvetenskap och teknik
Thomas Padron-McCarthy (thomas.padron-mccarthy@oru.se)

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

lördag 8 december 2012

Gäller som tentamen för:
DT3030 Datateknik C, Kompilatorer och interpretatorer, provkod 0100

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 37. För godkänt betyg (3 respektive G) krävs totalt minst 20 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast lördag 29 december 2012.
Återlämning av tentor:	Efter att resultatet meddelats kan tentorna hämtas på universitetets centrala tentamensutlämning.
Examinator och jourhavande:	Thomas Padron-McCarthy, telefon 070 - 73 47 013.

- Skriv tydligt och klart. Lösningar som inte går att läsa kan naturligtvis inte ge några poäng. Oklara och tvetydiga formuleringar kommer att misstolkas.
- Skriv den personliga tentamenskoden på varje inlämnat blad. Skriv *inte* namn eller personnummer på bladen.
- Skriv bara på en sida av papperet. Använd inte röd skrift.
- Antaganden utöver de som står i uppgifterna måste anges.
- Skriv gärna förklaringar om hur du tänkt. Även ett svar som är fel kan ge poäng, om det finns med en förklaring som visar att huvudtankarna var rätt.

LYCKA TILL!

Formelsamling

1. Eliminering av vänster-rekursion

En vänsterrekursiv grammatik kan skrivas om så att den inte är vänsterrekursiv. Antag att en regel (eller, korrektare uttryckt, två produktioner), i grammatiken ser ut så här:

```
A -> A x | y
```

A är en icke-terminal, men **x** och **y** står för godtyckliga konstruktioner som består av terminaler och icke-terminaler.

Regeln ersätts av följande två regler (eller, korrektare uttryckt, tre produktioner), som beskriver samma språk men som inte är vänsterrekursiva:

```
A -> y R
R -> x R | empty
```

2. Vänsterfaktorisering

Antag att grammatiken innehåller denna regel (två produktioner):

```
A -> x y | x z
```

A är en icke-terminal, men **x**, **y** och **z** står för godtyckliga konstruktioner som består av terminaler och icke-terminaler.

Skriv om till dessa tre produktioner:

```
A -> x R
R -> y | z
```

Uppgift 1 (10 p)

En kompilators arbete brukar delas in i sex eller sju faser. Ange vilka det är, och förklara kort vad varje fas gör. Vad är in- och utdata från respektive fas?

Uppgift 2 (6 p)

Det här är ett programavsnitt i ett C-liknande språk:

```
x = y;
z = 17 - t - u;
if (v + 2 == w + 19 * 2 - 3) {
    a = x;
    b = 2 * (a + c);
}
```

Översätt ovanstående programavsnitt till *två* av följande tre typer av mellankod.

a) ett abstrakt syntaxträd (genom att rita upp trädet!)

b) postfixkod för en stackmaskin

c) treadresskod

Observera: Det finns tre deluppgifter i uppgiften ovan. Välj ut och besvara (högst) *två* av dessa. (Skulle du svara på alla tre, räknas den med högst poäng bort.)

Uppgift 3 (7 p)

Förklara kort följande begrepp från kompilatortekniken.

a) aktiveringspost (på engelska: "activation record")
b) copy propagation
c) lexem
d) namnekvivalens (engelska: "name equivalence")
e) strukturekvivalens (engelska: "structural equivalence")
f) shift/reduce-konflikt
g) tvetydig grammatik

Scenario till de övriga uppgifterna

Vi vill skapa ett språk för att styra hushållsmaskiner. De maskiner som man ska kunna styra är **tvättmaskinen**, **diskmaskinen** och **spisen**. Man kan **starta** och **stänga av** maskinerna. Dessutom kan man **vänta på** att en maskin ska bli klar. Ett **styrprogram** består av ett eller flera sådana kommandon, separerade med semikolon. Exempel:

```
starta diskmaskinen;
starta  tvättmaskinen;
vänta    på diskmaskinen; stäng av diskmaskinen ; starta spisen;
vänta på

tvättmaskinen;
  stäng av tvättmaskinen ;
stäng av spisen
```

Uppgift 4 (2 p)

Ange vilka terminaler, dvs typer av tokens, som behövs för att man ska kunna skriva en grammatik för språket.

Uppgift 5 (4 p)

Skriv en grammatik för språket. Startsymbolen ska vara **styrprogram**.

Uppgift 6 (8 p)

Skriv en prediktiv recursive-descent-parser för språket, i ett språk som åtminstone liknar C, C++, C# eller Java. Du behöver inte skriva exakt korrekt programkod, men det ska framgå vilka procedurer som finns, hur de anropar varandra, och vilka jämförelser med tokentyper som görs. Du kan anta att det finns en funktion som heter **scan**, och att den returnerar typen på nästa token.

Om grammatiken från uppgiften ovan inte lämpar sig för implementation med en prediktiv recursive-descent-parser, visa först hur man gör om den på lämpligt sätt.
