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

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

måndag 19 augusti 2013

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 32. För godkänt betyg (3 respektive G) krävs totalt minst 19 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 9 september 2013.
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?

Scenario till de övriga uppgifterna

Vi vill skapa ett språk för att specificera vilka datorsalar vi har, och vilka datorer de innehåller. Man ska kunna ange att det finns datorsalar, till exempel **T120** och **T122**:

```
sal T120;
sal T122;
```

Man ska kunna ange att det finns datorer, och i vilken sal varje dator står:

```
dator t122-1 i sal T122;
dator t122-2 i sal T122;
dator stud114 i sal T120;
```

Varje kommando avslutas med ett semikolon. De kan skrivas i fritt format, dvs att man kan stoppa in extra mellanslag och radbrytningstecken på samma sätt som i vanliga språk som C och Java. Hela inmantningen avslutas med kommandot **klart**.

Exempel på en hel inmatning:

```
sal    T122   ;
       dator t122-1 i      sal T122;
dator t122-2

i sal T122;
sal T120; sal T124;
dator t122-3 i sal T122;
dator stud114 i sal T120; dator
stud115 i sal T120;        klart;
```

Tanken är att man sedan ska kunna söka efter datornamn och se var datorn finns, och efter salsnamn och se vilka datorer salen innehåller. Men här är det bara specifikationsspråket vi arbetar med.

Uppgift 2 (2 p)

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

Uppgift 3 (4 p)

Skriv en grammatik för språket. Startsymbolen ska vara **inmatning**, som representerar en hel inmatning som i exemplet ovan i scenariot.

Uppgift 4 (3 p)

Rita upp parse-trädet (även kallat konkret syntaxträd) för nedanstående inmatning, enligt grammatiken i uppgiften ovan:

```
sal vardagsrummet;
dator tv-datorn i vardagsrummet;
sal källaren;
klart;
```

Uppgift 5 (8 p)

Skriv en prediktiv recursive-descent-parser för datorspecifikationsspråket, 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 uppgift 3 inte lämpar sig för implementation med en prediktiv recursive-descent-parser, visa först hur man gör om den på lämpligt sätt.

Uppgift 6 (5 p)

Här är tio olika begrepp från kompilatortekniken. I vårt system för att specificera datorsalar och datorer, som innehåller språket ovan, kommer en del av dessa begrepp att vara relevanta (dvs, vi använder oss av de sakerna) och en del troligen inte. Ange för varje begrepp hur den saken används i vårt system, eller varför den inte behövs alls.

1. aktiveringspost (på engelska: "activation record")
2. basic block
3. Bison
4. Flex
5. lexikalisk analys
6. mark and sweep
7. postfixkod
8. strukturekvivalens (engelska: "structural equivalence"), som är motsatsen till namnekvivalens (engelska: "name equivalence")
9. symboltabell (engelska: "symbol table")
10. treadresskod
