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 26 oktober 2015

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 35. För godkänt betyg krävs totalt minst 20 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 16 november 2015.
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 vänsterrekursion

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 ett antal faser. Ange vilka det är, och förklara kort vad varje fas gör. Vad är in- och utdata från respektive fas?

Uppgift 2 (7 p)

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

```
x = 0;
while (x < y) {
    while (y < z) {
        y = y + z + 1;
    }
    z = 2;
}
```

Översätt ovanstående programavsnitt till var och en av följande tre typer av mellankod.

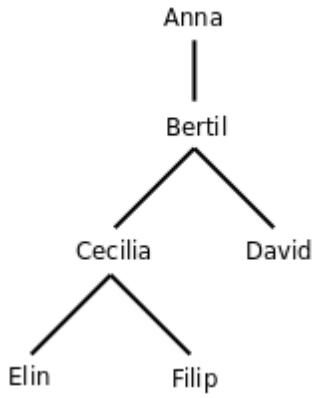
a) ett syntaxträd, även kallat abstrakt syntaxträd (genom att rita upp trädet!)

b) postfixkod för en stackmaskin

c) treadresskod

Scenario till de övriga uppgifterna

Här är ett (något förenklat) släktträd:



Anna är förälder till Bertil. Bertil är förälder till Cecilia och David. Cecilia är förälder till Elin och Filip.

Vi vill kunna specificera släktträdet genom att ange alla kopplingarna, dvs vem som är barn till vem, så här:

> **Bertil är barn till Anna. Cecilia är barn till Bertil. David är barn till Bertil. Elin är barn till Cecilia. Filip är barn till Cecilia. Klart!**

Man anger alltså ett antal kopplingar med **är barn till**. Varje koppling avslutas med punkt. Allra sist kommer ordet **Klart** och ett utropstecken.

Här är ytterligare tre exempel på hur inmatningen kan se ut:

> **Klas är barn till Sven. Klart!**
>
> **Klas är barn till Sven. Sven är barn till Svea. Klart!**
>
> **Nisse är barn till Nisse. Nisse är barn till Nisse. Nisse är barn till Nisse. Klart!**

Man ska kunna stoppa in blanktecken och radslut var som helst, utom inuti orden.

De här inmatningarna är *inte* tillåtna:

> **Klas är barn till Sven.**
>
> **Elin och Filip är barn till Cecilia. Klart!**

Uppgift 3 (2 p)

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

Uppgift 4 (4 p)

Skriv en grammatik för släktträdsinmatningar. Startsymbolen ska vara **släktträd**, som representerar ett enda släktträd enligt scenariot ovan.

Uppgift 5 (3 p)

Rita upp parse-trädet (även kallat konkret syntaxträd) för det här exemplet från scenariot, enligt din grammatik i uppgiften ovan:

> **Klas är barn till Sven. Sven är barn till Svea. Klart!**

Uppgift 6 (2 p)

Apropå exemplen från scenariot, så nog ser väl det här exemplet väldigt underligt ut?

> **Nisse är barn till Nisse. Nisse är barn till Nisse. Nisse är barn till Nisse. Klart!**

Nisse är barn till sig själv, flera gånger om!

Förklara hur det påverkar parsningen!

Uppgift 7 (7 p)

Skriv en prediktiv recursive-descent-parser för släktträdsspecifikationer, 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.
