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

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

onsdag 9 januari 2019

Gäller som tentamen för:
DT125G Kompilatorer och interpretatorer, provkod 0100

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 36. För godkänt betyg krävs totalt minst 18 poäng.
Resultat:	Meddelas på kursens hemsida eller via e-post senast onsdag 30 januari 2019.
Återlämning av tentor:	Elektroniskt via Studentforum.
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 (5 p)

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

```
while (x > y - z - t) {
    if (x < y)
        x = y + z * t;
    else
        x = y * z + t;
    x = y * (z + t);
    x = (y + z) * t;
}
```

Översätt ovanstående programavsnitt till *två* av följande tre typer av mellankod. (Skulle du svara på alla tre, räknas den med högst poäng bort.)

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

b) postfixkod för en stackmaskin

c) treadresskod

Scenario till uppgift 3-6

Här är en grantopp med några kottar:

Vi ska lagra data om skogar, träd och kottar. Kottarna har längd och vikt. Varje träd har en höjd och noll eller flera kottar. En skog innehåller ett eller flera träd. För att mata in data om skogen behöver vi ett särskilt skogsspråk, där en inmatning kan se ut så här:

```
skog {
    träd 17.9 {
        kotte 0.24 0.09;
        kotte 0.22 0.11;
        kotte 0.22 0.11;
    }
    träd 13 {
        kotte 0.26 0.10;
    }
    träd 13 { }
}
```

Denna inmatning beskriver en skog med tre träd: ett träd som är 17.9 meter högt och har tre kottar, ett träd som är 13 meter högt och har en kotte, och ett träd till som också är 13 meter högt, men som inte har några kottar alls. Den första av de angivna kottarna är 0.24 meter lång och väger 0.09 kilo.

Inmatningen ska kunna skrivas på fritt format, som de flesta vanliga programmeringsspråk, till exempel så här:

```
skog { träd 17.9 { kotte 0.24 0.09; kotte 0.22 0.11; kotte
0.22 0.11; } träd 13 { kotte 0.26 0.10; } träd 13 { } }
```

Uppgift 3 (3 p)

a) (1p) En av terminalerna, dvs typer av tokens, som behövs för att man ska kunna skriva en grammatik för skogsspråket är **tal**, som beskriver ett positivt reellt tal. Ett tal kan se ut på två sätt. Antingen kan det vara ett heltal, som helt enkelt består av en eller flera siffror. Ett exempel är **13**. Alternativt kan det se ut som ett tal med decimaler, dvs en eller flera siffror, följt av en punkt, följt av en eller flera siffror. Ett exempel är **17.9**. Skriv ett reguljärt uttryck som beskriver hur ett tal får se ut.

b) (2p) Vilka andra terminaler, förutom **tal**, behövs för att man ska kunna skriva en grammatik för språket?

Uppgift 4 (4 p)

Skriv en grammatik för skogsspråket. Startsymbolen ska vara **skogsbeskrivning**, som representerar en inmatning av en skog enligt scenariot ovan.

Uppgift 5 (6 p)

a) (3p) Ett parse-träd (ibland kallat "konkret syntaxträd") innehåller noder för alla icke-terminaler. Rita upp parse-trädet för den här inmatningen, enligt din grammatik i uppgiften ovan:

```
skog {
    träd 7 {
        kotte 0.24 0.09;
    }
}
```

b) (1p) Hur skiljer sig ett syntaxträd (ibland kallat "abstrakt syntaxträd") från ett parse-träd?

c) (2p) Rita upp syntaxträdet för inmatningen ovan!

Uppgift 6 (8 p)

Skriv en prediktiv recursive-descent-parser för skogsspråket, 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**, som returnerar typen på nästa token, och en funktion som heter **error**, som man kan anropa när något gått fel och som skriver ut ett felmeddelande och avslutar programmet.
