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

Tentamen i

Kompilatorer och interpretatorer

måndag 13 januari 2020

Gäller som tentamen för:
DT501A Kompilatorer och interpretatorer för civilingenjörer, provkod A001

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 38. För godkänt betyg krävs totalt minst 19 poäng.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 3 februari 2020.
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 (8 p)

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

```
a = b - 2 * c - 3;
d = 1;
while (e * 4 < f * 5) {
    g = g + 6;
    h = j + 7;
    k = g + 6;
    m = j + 7;
}
```

a) Översätt ovanstående programavsnitt till *antingen* ett abstrakt syntaxträd (genom att rita upp trädet) *eller* postfixkod för en stackmaskin. (Inte båda!)

b) Översätt programavsnittet till treadresskod. Identifiera vilka basic blocks som finns, och rita upp flödesgrafen. (Flödesgrafen, med treadresskoden i, räcker som svar.)

c) Visa någon optimering som går att göra inom ett av dessa basic blocks.

Scenario till uppgift 3-6

SQL är ett språk för att arbeta med databaser, till exempel för att söka i en databas. Man söker med kommandon som inleds med nyckelordet **select**.

Vi ska arbeta med en delmängd av SQL som kan uttrycka sökningar, eller "SQL-frågor", som ser ut så här:

- Först skriver man nyckelordet **select**.
- Därefter kommer ett eller flera namn, åtskilda med kommatecken, som är namn på kolumner i tabeller.
- Därefter kommer nyckelordet **from**, följt av ett eller flera namn, åtskilda med kommatecken, som är namn på tabeller.
- Därefter är frågan antingen slut, eller så kommer nyckelordet **where**, följt av ett villkor.
- Villkoret består av ett eller flera delvillkor, somm sammanbinds med nyckelordet **and**.
- Ett delvillkor är en jämförelse av något slag. Det består av två operander, som kan vara antingen konstanter eller namn på kolumner, som jämförs med en jämförelseoperator.
- En konstant är antingen ett heltal, som **17**, eller en textsträng med enkla citationstecken, som **'Kalle'**.
- Jämförelsen görs med någon av de här jämförelseoperatorerna:
  = != < <= > >=
- Hela SQL-frågan avslutas med ett semikolon (**;**).

Här är några exempel på SQL-frågor som man ska kunna skriva:

```
select Nummer from Gurkor where Vikt > 3;

select Nummer, Vikt from Gurkor where Vikt > 3 and Bredd <= 3;

select Kod, Maxlast, Bruttovikt
from Vagnar, Bilar, Personer
where Vagninnehavare = Personnummer
and Bilinnehavare = Personnummer
and Namn = 'Thomas';

select Nummer, Nummer, Nummer
from Gurkor, Gurkor, Gurkor
where Vikt != Vikt and 17 < 17;
```

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

```
     select
Nummer from
     Gurkor where Vikt

   > 3;
```

Uppgift 3 (4 p)

a) (1p) En av terminalerna, dvs typer av tokens, som behövs för att man ska kunna skriva en grammatik för SQL-delmängden är **namn**, som är ett namn på en kolumn eller en tabell. Det är helt enkelt en eller flera stora eller små bokstäver, blandade hur som helst, till exempel **Gurkor**, **ViktAntal** eller **strutsKamelX**. (Vi kan nöja oss med det engelska alfabetet med bokstäverna A-Z.) Skriv ett reguljärt uttryck som beskriver hur ett namn får se ut.

b) (1p) En annan av terminalerna som behövs är **text**, som är en textsträng med enkla citationstecken, som **'Kalle'**. Innanför citationstecknen kan det finnas vilka tecken som helst utom just de citationstecknen. Skriv ett reguljärt uttryck som beskriver hur en textsträng får se ut.

c) (2p) Vilka andra terminaler, förutom **namn** och **textsträng**, behövs för att man ska kunna skriva en grammatik för SQL-delmängden?

Uppgift 4 (5 p)

Skriv en grammatik för SQL-delmängden. Startsymbolen ska vara **kommando**, som representerar en enda SQL-fråga enligt scenariot ovan.

Uppgift 5 (3 p)

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

```
select Nr from Bilar, Fiskar where Kod = 'X7';
```

Uppgift 6 (8 p)

Skriv en prediktiv recursive-descent-parser för SQL-delmängden. 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.
