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 4 november 2013

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 46. För godkänt betyg krävs totalt minst 26 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 25 november 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 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 (3 p)

Det är meningen att följande C-program ska skriva ut **Hej hopp världen!**
När vi "bygger" (dvs kompilerar och länkar) programmet, får vi de angivna kursiverade fel- och varningsmeddelandena. I vilken av kompilatorns faser (eller på annan plats) upptäcks vart och ett av dessa fel och varningar?

Program	Fel och varningar
`#include <stdiå.h>`	*stdiå.h: No such file or directory*
`int maig(void) {`	*undefined reference to `main'*
`    printf("Hej ");`	
`    printf("hopp );`	*error: missing terminating " character*
`    printf printf("världen!\n");`	*error: expected ';' before 'printf'*
`    return "Hej!";`	*warning: return makes integer from pointer*
`}`	

Uppgift 3 (7 p)

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

```
b = a;
while (a < d) {
    c = c * 2 + a;
    a = a - b + 2 * 3;
}
```

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

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

b) postfixkod för en stackmaskin

c) treadresskod

Uppgift 4 (4 p)

Ange vilka basic blocks som mellankoden från deluppgift c i uppgiften ovan innehåller. Optimera därefter koden med hjälp av de vanliga metoderna för optimering av treadresskod.

Uppgift 5 (3 p)

Jämför mark and sweep med referensräkning. Vilja fördelar och nackdelar har de?

Scenario till de övriga uppgifterna

Försvarsmakten behöver hålla reda på sina tältkaminer.

Vi vill skapa ett språk för att specificera vilka kaminer de har, och vilket skick de är i. Varje kamin har ett unikt nummer, och i språket ska man kunna ange att det finns en kamin, till exempel så här för kamin nummer **723**:

```
kamin 723;
```

Man ska kunna ange vilket skick kaminen är i:

```
skick 723 usel;
```

De olika skick som finns ska skapas med kommandot **skapa skick**:

```
skapa skick usel;
```

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 inmatningen avslutas med kommandot **klart**.

Exempel på en hel inmatning:

```
kamin 84;
    kamin   232    ;
skapa skick bra;
 skapa
  skick
   trasig
    ;skapa skick kasserad;
skick 84 bra;
skick 84 trasig;
skick 232 bra;
skick 84 bra;
skick 84 kasserad;
klart;
```

Tanken är att man senare ska kunna skriva ut listor över kaminer och deras skick. Men här är det bara inmatningsspråket vi arbetar med.

Uppgift 6 (3 p)

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

b) Ange reguljära uttryck för de terminaler som inte har ett fast utseende.

Uppgift 7 (4 p)

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

Uppgift 8 (3 p)

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

```
skapa skick skrattretande;
kamin 1;
skick 2 absurd;
klart;
```

Uppgift 9 (2 p)

I uppgiften ovan anger man i inmatningen att kamin 2 har skicket "absurd", Men varken den kaminen eller det skicket har definierats tidigare. Förklara hur det påverkar parsningen!

Uppgift 10 (7 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.
