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

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

lördag 7 december 2013

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 43. För godkänt betyg krävs totalt minst 24 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast lördag 28 december 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)

När vi "bygger" (dvs kompilerar och länkar) och till sist (efter att ha rättat kompileringsfelen) provkör följande C-program, får vi de angivna 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 <stdio.h>`
 	
`int main(void) {`
`  int ena, andra;`
`  int *pekaren = NULL;`
 	
`  prontf("Hej!\n");`
`  printf("Ange ett tal: ";`
`  scanf("%d", &ena);`
`  printf("Ange ett tal till: ");`
`  scanf("%d", @andra);`
`  printf("Och ett tredje: ");`
`  scanf("%d", &tredje);`
`  pekaren = ena;`
`  *pekaren = andra;`
 	
`  return 0;`
`}`
	`implicit declaration of function 'prontf'`
	`expected ')' before ';' token`
	
	`unknown character: '@'`
	
	`'tredje' undeclared`
	`assignment makes pointer from integer`
	`General Protection Fault`

Uppgift 3 (7 p)

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

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

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 (2 p)

I uppgiften ovan innehåller loopvillkoret variablerna **z** och **y**. Ingen av dessa ändras i kroppen på loopen. Om loopvillkoret är sant från början, hamnar programmet i en oändlig loop.

Skulle någon eller några faser av kompilatorn kunna påverkas av att loopen är oändlig? Vilka i så fall, och hur påverkas de?

Uppgift 5 (4 p)

Här är ett C-program. Ett programs adressrymd kan delas upp i fyra delar: programkod och konstanter, statiska data, heap och stack. Rita upp hur stacken (med aktiveringsposter), heapen och statiska data ser ut när programkörningen kommer till utskriften av "Här!".

```
#include <stdio.h>

int alpha = 1;

int bravo(void) {
    alpha = 10;
    printf("Här!\n");
    return 1;
}

int charlie(int alpha) {
    if (alpha <= 0)
        return bravo();
    else
        return alpha * charlie(alpha - 1);
}

int main(void) {
    int delta = 0;
    delta = charlie(3);
    printf("delta = %d\n", delta);
    return 0;
}
```

Scenario till de övriga uppgifterna

En relationsdatabas är en databas där man lagrar data i tabeller. Med språket SQL kan man skapa nya tabeller med kommandot **create table**. Vi ska implementera en förenklad version av det kommandot.

Här är ett exempel på ett kommando som skapar tabellen **maskar**, som har de tre kolumnerna **nummer**, **namn** och **bor_i**. Kolumnen **nummer** är så kallad primärnyckel i tabellen.

```
create table maskar
(nummer integer primary key,
namn char(10),
bor_i integer)
```

Här är ett annat exempel, ett kommando som skapar tabellen **svampar**, som har de fyra kolumnerna **artnamn**, **vikt**, **svamp_id** och **smak**.

```
create table svampar
(artnamn char(30),
vikt integer,
svamp_id integer,
smak char(20))
```

Varje kommando börjar med nyckelorden **create** och **table**, och sen kommer, inom parenteser, de kolumner som tabellen ska ha.

Varje kolumnspecifikation börjar med ett namn, följt av en datatyp som är antingen **integer** eller **char(*n*)**, där ***n*** är ett heltal som anger antalet tecken som får plats i kolumnen.

En kolumn kan dessutom anges som primärnyckel, genom att man skriver nyckelorden **primary** och **key** som i det första exemplet.

Kommandot 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.

Ännu ett exempel:

```
create table
   bilar  (regnummer char(6)
,
       modellnamn char(  100           ),
id integer primary     key,
          pris integer)
```

Tanken är att man senare ska kunna skapa mer komplicerade tabeller, och även lägga in data och söka. Men här är det bara det förenklade **create table**-kommandot 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 **kommando**, som representerar ett enda **create table**-kommando enligt scenariot ovan.

Uppgift 8 (3 p)

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

```
create table kaffekoppar
(nummer integer primary key,
beskrivning char(100))
```

Uppgift 9 (7 p)

Skriv en prediktiv recursive-descent-parser för **create table**-kommandot, 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.
