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

Tentamen i

Kompilatorer och interpretatorer (*nya* kursen)

för Dataingenjörsprogrammet m fl

måndag 5 november 2012

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 35. För godkänt betyg (3 respektive G) krävs totalt minst 20 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 26 november 2012.
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 vänster-rekursion

En vänsterrekursiv grammatik kan skrivas om så att den inte är vänsterrekursiv. Antag att en regel i grammatiken ser ut så här:

```
A -> A x | y
```

A är en icke-terminal, men **x** och **y** kan stå för godtyckliga konstruktioner, som innehåller en eller flera terminaler och icke-terminaler.

Regeln ersätts av följande två regler, som beskriver samma språk men som inte är vänsterrekursiva:

```
A -> y R
R -> x R | empty
```

2. Vänsterfaktorisering

Antag att grammatiken innehåller dessa två (ja, två) produktioner:

```
A -> x y | x z
```

A är en icke-terminal, men **x** och **y** kan stå för godtyckliga konstruktioner, som innehåller en eller flera terminaler och icke-terminaler.

Skriv om till dessa tre (ja, tre) produktioner:

```
A -> x R
R -> y | z
```

Uppgift 1 (10 p)

En kompilators arbete brukar delas in i sex eller sju faser. Ange vilka det är, och förklara kort vad varje fas gör. Vad är in- och utdata från respektive fas?

Uppgift 2 (6 p)

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

```
a = b;
c = 17;
while (x < 19) {
    x = x + b - c - 1;
    if (c > 0)
        c = c - 1;
    r = r + 2 * b;
}
```

Översätt ovanstående programavsnitt till *två* av följande tre typer av mellankod.

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

b) postfixkod för en stackmaskin

c) treadresskod

Observera: Det finns tre deluppgifter i uppgiften ovan. Välj ut och besvara (högst) *två* av dessa. (Skulle du svara på alla tre, räknas den med högst poäng bort.)

Uppgift 3 (10 p)

Vi vill skapa ett språk för att kunna hantera anslag och anslagstavlor. Här är en grammatik för språket. Terminaler skrivs med **fetstil**, och icke-terminaler *kursiverade*. ***tomt*** står för en tom följd av terminaler och icke-terminaler.

kommandolista -> *kommando kommandolista* | ***tomt***
kommando -> *anslagskommando* | *anslagstavlekommando* | *placeringskommando*
anslagskommando -> **skapa anslag heltal** *rubrik innehåll*
rubrik -> **textsträng**
innehåll -> **textsträng**
anslagstavlekommando -> **skapa anslagstavla heltal**
placeringskommando -> **sätt upp anslag heltal på anslagstavla heltal**

Här är också en Flex-fil som beskriver hur terminalerna i språket ser ut:

```
%{
    #include "anslag.tab.h"
%}
%%
[\n\t ] { /* Ignorera whitespace */ }
skapa { return skapa; }
anslag { return anslag; }
anslagstavla { return anslagstavla; }
sätt { return sätt; }
upp { return upp; }
på { return på; }
[0-9]+ { return heltal; }
\"[^"]*\" { return textsträng; }
. { fprintf(stderr, "Ignorerar felaktigt tecken: '%c'\n", *yytext); }
%%
```

a) Icke-terminalen *kommando* beskriver de kommandon som man kan ge. Ange fem olika kommandon som ingår i det språk som beskrivs av grammatiken ovan. Variera dem så att de inte är alltför lika.

b) Ange fem olika kommandon som *inte* ingår i det språk som beskrivs av grammatiken ovan, men som man kanske skulle kunna tro gjorde det om man inte var så bra på grammatiker och Flex.

c) Den här grammatiken lämpar sig inte för implementation av en prediktiv recursive-descent-parser. Ange *vilket eller vilka* problem som finns, *vad* i grammatiken som orsakar problemen, och *vad som händer* i parsern.

d) Använd de transformationer som tagits upp i kursen för att omvandla grammatiken till en grammatik som beskriver samma språk, men som enkelt kan implementeras i en prediktiv recursive-descent-parser. Ange vilka transformationer du gör, och visa vad resultatet blir för den här grammatiken.

e) Skriv den prediktiva recursive-descent-parsern i ett språk som åtminstone liknar C, Java eller Pascal. 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.

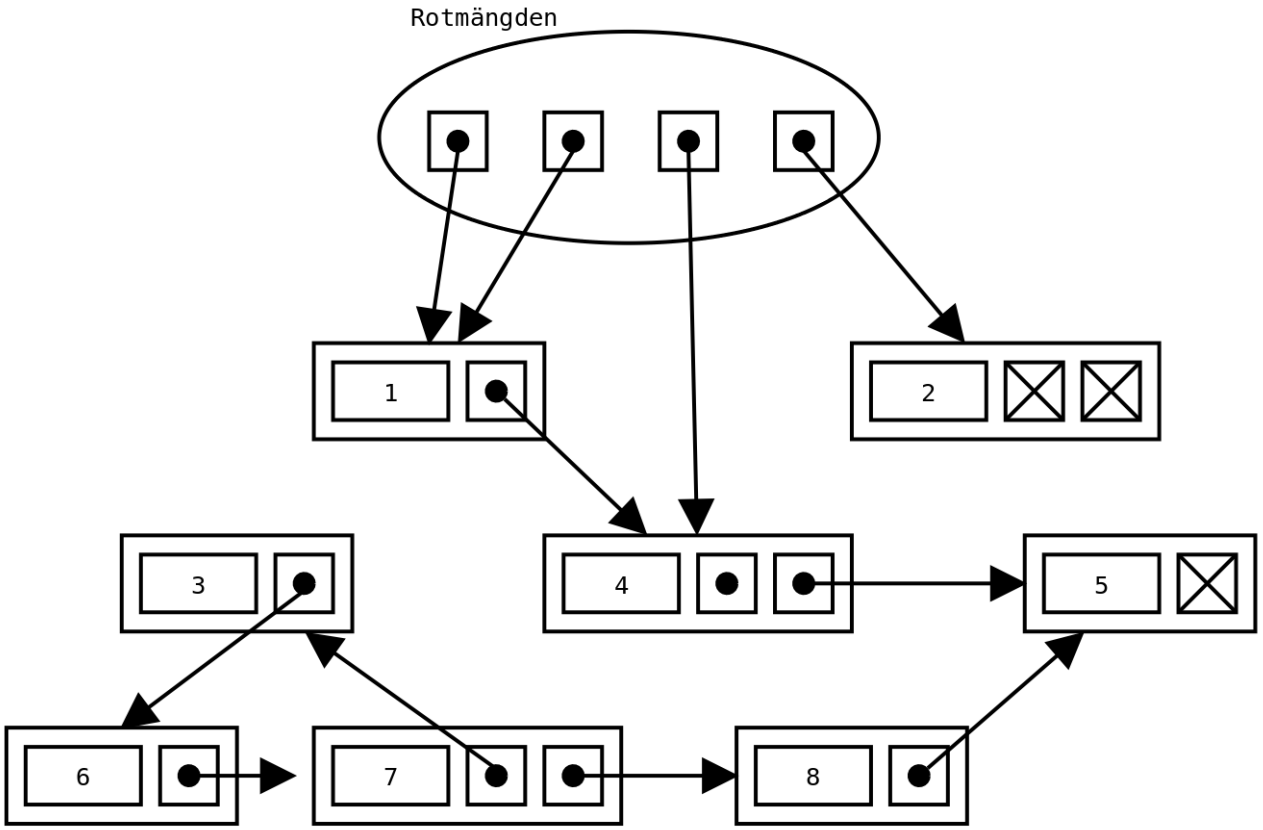
Uppgift 4 (4 p)

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 x = 1, y = 2;

int f(int x, int y) {
    printf("Här!\n");
    return x + y;
}

int g(int x) {
    if (x < 5)
        return g(x + 1);
    else
        return f(x, y);
}

int main(void) {
    g(3);
    return 0;
}
```

Uppgift 5 (5 p)

Antag att minnet innehåller dessa dataobjekt.



a) Vi använder skräpsamlingsmetoden "mark and sweep". Vilka av objekten kommer att markeras i markeringsfasen? Vad händer med de objekt som markerats? Vad händer med de objekt som inte markerats?

b) Vad har rotmängden för funktion?

c) Antag att vi i stället använder referensräkning. Ange värdet på referensräknaren för vart och ett av dataobjekten.
