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 24 oktober 2016

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 41. 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 måndag 14 november 2016.
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)

När vi kompilerar, länkar och sedan kör följande försök till C-program, får vi de kursiverade fel- och varningsmeddelandena. I vilka faser, eller vid andra tidpunkter, upptäcks vart och ett av de olika felen och varningarna? (Vi kan behöva rätta en del av felen för att komma vidare så att de andra felen upptäcks.)

```
#include <stdi.h> // fatal error: stdi.h: No such file or directory

int main(void) {
    double d1 = 1.0; // warning: unused variable 'd1'
    int d1 = 2; // error: conflicting types for 'd1'
    double 2; // error: expected identifier or '(' before numeric constant
    int i = 17;

    printf(Hej!); // error: 'Hej' undeclared
                  // error: expected ')' before '!' token

    printg("%d", i); // undefined reference to `printg'
    printf("%s", i); // Segmentation fault (core dumped)

    return 0;
}
```

Uppgift 3 (5 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, med innehåll, när programkörningen kommer till utskriften med printf.

```
#include <stdlib.h>
#include <stdio.h>

int a = 1, b = 2;

int f(int x, int y) {
    int c;
    int *d;
    a = x;
    c = 3;
    d = malloc(sizeof(int));
    *d = b;
    if (x < 2) {
        return x * f(x + 1, y + 2);
    }
    else {
        printf("Nu!\n");
        return 1;
    }
}

int main(void) {
    int a;
    int *b;
    a = 2;
    b = malloc(sizeof(int));
    *b = 3;
    a = f(0, 4);
    return 0;
}
```

Uppgift 4 (5 p)

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

```
a = 1;
b = c * 2 + 3 + d;
if (a < b) {
    while (a + 4 < b * 5 + c) {
        a = a + 6;
    }
}
else {
    a = 7;
    b = 8;
}
```

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

a) ett syntaxträd, även kallat 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.)

Scenario till de övriga uppgifterna

Vi ska bygga ett system för att beställa fikabröd från ett bageri, med en klientdel där man kryssar för i rutor, och en serverdel som hanterar beställningarna. När klientdelen skickar en beställning till serverdelen följer kommunikationen ett särskilt protokoll.

En beställning börjar med nyckelordet **start** och avslutas med nyckelordet **klart**. Efter nyckelordet **start** kommer nyckelordet **namn** följt av en textsträng som anger namnet på kunden. Därefter kan det, men måste inte, komma nyckelordet **telefon** följt av en textsträng som anger telefonnumret till kunden. Efter detta en eller flera fikabrödsbeställningar, där var och en består av ett tal (som kan innehålla decimaler) och ett av orden **kanelbulle**, **wienerbröd** och **chokladboll**. Blanktecken, som mellanslag och radslut, ska ignoreras, förutom inuti nyckelord, tal och textsträngar.

Här följer två exempel på beställningar:

```
start
namn "Anna A. Andersson"
telefon "070-73 47 013"
1 kanelbulle
2 wienerbröd
1 kanelbulle
1 chokladboll
3 chokladboll
klart
```

```
          start namn
        "Anna A. Andersson" 1
kanelbulle 2      wienerbröd



                klart
```

Uppgift 5 (2 p)

Ange vilka terminaler, dvs typer av tokens, som behövs för att man ska kunna skriva en grammatik för fikabeställningarna i scenariot.

Uppgift 6 (4 p)

Skriv en grammatik för fikabeställningar. Startsymbolen ska vara **beställning**, som representerar en komplett beställning enligt scenariot ovan.

Uppgift 7 (3 p)

Rita upp parse-trädet (även kallat konkret syntaxträd) för den här beställningen, enligt din grammatik i uppgiften ovan:

```
start
namn "Olle"
1 kanelbulle
2 wienerbröd
klart
```

Uppgift 8 (7 p)

Skriv en prediktiv recursive-descent-parser för fikabeställningar, 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.
