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

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

fredag 8 januari 2016

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 36. För godkänt betyg krävs totalt minst 21 poäng, varav minst 8 poäng på uppgift 1.
Resultat:	Meddelas på kursens hemsida eller via e-post senast fredag 29 januari 2016.
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 dessa 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 (6 p)

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

```
a = 0;
b = 14;
if (a == c) {
    while (a != d) {
        a = c - d - 2;
        c = c + 1;
    }
    d = 4;
}
else {
    c = c * 2 + 1;
    d = 3;
}
```

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.)

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

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

int a = 10, b = 20;

int g(int a) {
    int z;
    int *p;
    if (a == 0)
        return 1000;
    z = a - 1;
    p = malloc(sizeof(int));
    *p = 2000;
    printf("z = %d\n", z);
    z = g(z);
    return z;
}

int f(int x, int y) {
    int z;
    int *p;
    x = x + y;
    z = 100;
    p = malloc(sizeof(int));
    *p = 200;
    a = g(x);
    return a;
}

int main(void) {
    int a, c;
    a = 1;
    b = 2;
    c = 3;

    a = f(a, b);
    return 0;
}
```

Uppgift 4 (5 p)

En hälsning kan, antar vi, vara antingen "hej" eller "hej på dig". Här är en grammatik för hälsningar:

hälsning -> **hej** | **hej på dig**

a)

Här är ett utdrag ur en prediktiv recursive-descent-parser, skriven i C, för hälsningar:

```
void halsning() {
    if (lookahead == HEJ) {
        match(HEJ);
    }
    else if (lookahead == HEJ) {
        match(HEJ); match(PA); match(DIG);
    }
    else {
        error();
    }
}
```

Det finns ett problem med den parsern. Förklara vad problemet är, och hur man kan lösa det.

b)

Här är ett utdrag ur en Bison-fil för att skapa en parser för samma hälsningar som ovan:

```
halsning : HEJ | HEJ PA DIG ;
```

Finns samma problem här som med recursive-descent-parsern ovan? Varför, eller varför inte?

Scenario till de övriga uppgifterna

Du hittar en prediktiv recursive-descent-parser, skriven i C, som har källkoden nedan.

```
#include <stdlib.h>
#include <stdio.h>
#include "tokenkoder.h"
// Ovanstående ger tokenkoderna:
// STARTA PA GA METER I RIKTNING TAL TILL KLART

extern int scan();

int lookahead;

void error() {
    printf("Det har tyvärr uppstått ett fel.\n");
    exit(EXIT_FAILURE);
}

void match(int expected) {
    if (lookahead != expected)
        error();
    else
        lookahead = scan();
}

void program(), startkommando(), gakommandon(),
     gakommando(), vart();

void program() {
    startkommando(); gakommandon(); match(KLART);
}

void startkommando() {
    match(STARTA); match(PA); match(TAL); match(TAL);
}

void gakommandon() {
    if (lookahead == GA) {
        gakommando(); gakommandon();
    }
    else {
        /* tomt */
    }
}

void gakommando() {
    match(GA); vart();
}

void vart() {
    if (lookahead == TAL) {
        match(TAL); match(METER); match(I);
        match(RIKTNING); match(TAL);
    }
```

```
    else if (lookahead == TILL) {
        match(TILL); match(TAL); match(TAL);
    }
    else if (lookahead == GA) {
        match(GA); match(GA);
    }
    else if (lookahead == PA) {
        match(PA);
    }
    else {
        error();
    }
}

void parse() {
    lookahead = scan();
    program();
    printf("Ok.\n");
}

int main() {
    printf("Mata in inmatningen.\n");
    printf("Avsluta med \"klart\" och EOF.\n");
    parse();
}

/************Inte med!*************/

// Behövs för Lex
int yywrap() {
    return 1;
}
```

Du hittar också en scannerspecifikation för Flex, som hör till parsern ovan, och som ser ut enligt nedan:

```
%{
    #include "tokenkoder.h"
%}

%%

[\n\t ] { /* Ignorera whitespace */ }
"starta"        { return STARTA; }
"på"            { return PA; }
"gå"            { return GA; }
"meter"         { return METER; }
"i"             { return I; }
"riktning"      { return RIKTNING; }
"till"          { return TILL; }
"klart"         { return KLART; }
[0-9]+(\.[0-9]+)? { return TAL; }
. { fprintf(stderr, "Ignorerar felaktigt tecken: '%c'\n", *yytext

%%

int scan() {
```

```
        return yylex();
    }
```

Uppgift 5 (4 p)

Ange grammatiken för det språk som parsern i scenariot förstår.

Uppgift 6 (4 p)

Ange en tillåten inmatning för språket som parsern i scenariot förstår. Försök göra ett exempel som tydligt visar hur språket ser ut.

Uppgift 7 (3 p)

Ange en tillåten inmatning för språket som parsern i scenariot förstår. Det kan vara samma inmatning som i uppgiften ovan, eller en mycket enklare. Rita sedan upp parse-trädet (även kallat konkret syntaxträd) för inmatningen.
