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 23 oktober 2017

Gäller som tentamen för:
DT125G Kompilatorer och interpretatorer, provkod 0100
DT3030 Datateknik C, Kompilatorer och interpretatorer, provkod 0100

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 51. För godkänt betyg krävs totalt minst 30 poäng, varav minst 15 poäng på uppgift 8.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 13 november 2017.
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 (5 p)

Betrakta följande C-program:

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

int a, b, c;

void f(int d) {
    int a;
    int* p;

    a = 1;
    b = 2;
    c = 3;
    p = malloc(sizeof(int));
    *p = d;

    if (d < 1)
        f(d + 1);

    printf("Här!\n");
}

int main(void) {
    int a;
    int* p;

    a = 4;
    b = 5;
    c = 6;
    p = malloc(sizeof(int));
    *p = 8;
    f(0);
}
```

När programexekveringen för första gången kommer fram till raden med utskriften "Här!", så antar vi att vi stoppar programmet och tittar på vilka olika variabler som finns i minnet.

I minnet kommer det då att finnas ett antal heltalsvariabler och andra lagringsutrymmen för heltal, som innehåller olika tal. Rita upp en skiss över dessa variabler och lagringsutrymmen, med innehåll, och förklara vad som är vad. Din förklaring bör bland annat innehålla termerna "statiska data", "stack", "heap", "aktiveringspost" och "parametrar".

Uppgift 2 (5 p)

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

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

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

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

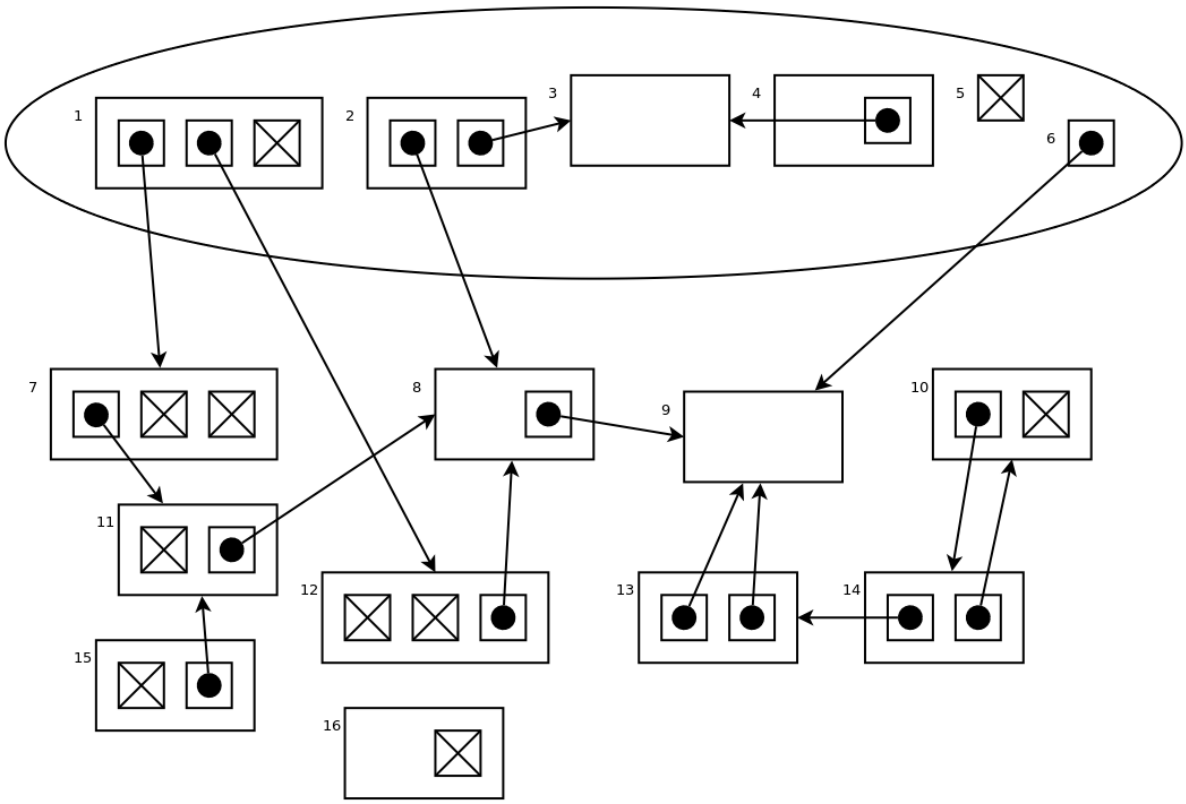
b) postfixkod för en stackmaskin

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 (3 p)

Antag att minnet innehåller dessa sexton dataobjekt. Pilarna är pekare, och kryss betyder NULL-pekare. De inringade objekten, nummer 1-6, utgör rotmängden.



a) Vi använder referensräkning. Ange värdet på referensräknaren för vart och ett av dataobjekten.

b) Vilka av dataobjekten kommer att tas bort?

c) Antag att vi i stället använder skräpsamling med mark-and-sweep. Vilka av dataobjekten kommer att tas bort av skräpsamlaren?

Scenario till uppgift 4-7

Vi har ett varuhus, och eftersom vi tycker att det är för lätt att slå in varor i de vanliga kassaregistren, ska vi byta till ett system där man skriver in vilka varor man handlar, på ett tangentbord.

Inmatningsspråket består av något vi kan kalla "köp", som består av nyckelordet **start**, följt av noll eller flera varurader, följt av nyckelordet **klart**. En varurad består av ett tal (eventuellt med en decimaldel, som då avskiljs med decimalkomma), följt av en enhet (till exempel **liter**) som kan utelämnas, följt av en textsträng som anger vilken vara man köpt. Exempel på en komplett inmatning av ett köp:

```
start
1 liter "mjölk"
2,3 kg "ekologiska morötter"
1 "apelsin"
19 "Coca-Cola 33 cl"
klart
```

Uppgift 4 (3 p)

a) (1p) En av terminalerna, dvs typer av tokens, som behövs för att man ska kunna skriva en grammatik för köpen är **textsträng**. En textsträng inleds och avslutas med ett citationstecken (**"**), och den kan innehålla bokstäver, siffror, bindestreck och mellanslag. Skriv ett reguljärt uttryck som beskriver hur en textsträng får se ut.

b) (2p) Vilka andra terminaler, förutom **textsträng**, behövs för att man ska kunna skriva en grammatik för köpen?

Uppgift 5 (4 p)

Skriv en grammatik för ett köp. Startsymbolen ska vara **köp**, som representerar en inmatning enligt scenariot ovan.

Uppgift 6 (3 p)

Ett parse-träd (ibland kallat "konkret syntaxträd") innehåller noder för alla icke-terminaler. Rita upp parse-trädet för det här köpet, enligt din grammatik i uppgiften ovan:

```
start
1 "apelsin"
2 "äpple"
klart
```

Uppgift 7 (8 p)

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

Denna sida avsiktligt lämnad tom.

Uppgift 8 (20 p)

Ange för varje påstående om det är sant eller falskt! Lämna in sidorna med den här uppgiften tillsammans med resten av svaren. Fel svar ger inte minuspoäng.

Påstående	Sant	Falskt
Den första fasen i kompilatorn är den lexikaliska analysatorn, även kallad scanner.		
Scannern gör om en sekvens av tokens till en sekvens av tecken.		
Den andra fasen i kompilatorn är den syntaktiska analysatorn, även kallad parser.		
I de flesta vanliga programmeringsspråk tolkas de tre tecknen **123** som ett heltal, vilket är en typ av token. För heltalet **123** är lexemet de tre tecknen 1, 2 och 3.		
För heltalet 123 är det lexikaliska värdet talet 123.		
Parsern analyserar programmet enligt källspråkets grammatik.		
Utdata från parsern är ofta ett syntaxträd, men det är inte säkert att man faktiskt bygger upp trädet, med pekare i minnet eller på annat sätt.		
Semantisk analys analyserar programmets betydelse, dvs om det stämmer med källspråkets grammatik.		
Det är den semantiska analysatorn som avgör vad additionen i deluttrycket **x+y** egentligen innebär, till exempel om det är addition av heltal eller om det är sammanslagning av strängar.		
Det är den semantiska analysen som lägger in symboler i symboltabellen.		
Mellankodsgeneratorn genererar mellankod ("intermediärkod"). Mellankoden skickas vidare till nästa fas som textrader, till exempel **t2 = x + t1**, och så körs scannern på nytt för att läsa in dem, men de är så enkla att parsern inte behövs.		
Maskinoberoende kodoptimering gör programmet mindre och snabbare (eller en av dessa). Det kan hända att det optimerade programmet blir större än utan optimering, till exempel med looputrullning.		
Kodgeneratorn genererar målkoden, som till exempel kan bestå av maskininstruktioner för en fysisk processor.		

Påstående	Sant	Falskt
Målkoden kan vara i form av text, till exempel assemblerinstruktioner som **ADD R1, R2**.		
Den maskinberoende optimeraren gör optimeringar som bara kan göras på målkodsnivå, till exempel sammanslagning av assemblerinstruktioner.		
Om scannern är implementerad i form av en funktion som heter **scan**, och parsern är implementerad i form av en funktion som heter **parse**, kommer funktionen **scan** alltid att anropas före funktionen **parse**.		
Den semantiska analysen resulterar i mellankod, som den maskinoberoende kodoptimeraren sen omvandlar till ett syntaxträd.		
Kodoptimeraren i vanliga programmeringsspråk som C och C# kan omvandla 1+2 till 3.		
Kodoptimeraren i vanliga programmeringsspråk som C och C# kan ibland omvandla en loop till ett konstant värde.		
Kodoptimeraren i vanliga programmeringsspråk som C och C# kan omvandla en länkad lista till en hashtabell.		