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

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

måndag 15 augusti 2011

Gäller som tentamen för:
DT3004 Datateknik C, Kompilatorer och interpretatorer, provkod 0100

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 30. För godkänt betyg (3 respektive G) krävs 15 poäng.
Resultat:	Meddelas på kursens hemsida eller via e-post senast måndag 5 september 2011.
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!

Uppgift 1: Pass och faser (6 p)

a) Vad är det, i kompilatorsammanhang, för skillnad på en fas och ett pass?

b) Vilka faser brukar man tala om? Skriv inte bara vad de heter, utan beskriv (mycket) kort vad de gör.

c) När vi "bygger" (dvs kompilerar och länkar) följande försök till C-program, får vi de tre kursiverade fel- och varningsmeddelandena. I vilka faser upptäcks dessa fel och varningar?

```
#include <stdio.h>

int main(void) {
    int a;
    int 2a;                     error: invalid suffix "a" on integer constant

    printf("Hej!\n");
    printg("Ange ett tal: ");   warning: implicit declaration of function 'printg'
                                error: undefined reference to `printg'
    scanf("%d", &a);
    printf("Talet = %d\n", a);

    return 0;
}
```

Uppgift 2: Mellankod (6 p)

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

```
a = 1;
b = a - b - c * d;
while (i * 2 > j * 3) {
    if (i % 2 == 0)
         i = i + 2;
    b = i;
}
```

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.)

Scenario till (en del av) uppgifterna

I en kartdatabas vill vi kunna lägga in tätorter och vägar mellan dem. Vi vill också kunna lägga in rutter, som leder från en tätort till en annan och som består av en eller flera vägar. För att mata in detta konstruerar vi ett enkelt språk.

Man använder nyckelordet **ort** för att lägga in en tätort, till exempel så här: **ort Örebro**

Man använder nyckelordet **väg** för att lägga in en väg, till exempel så här för att ange att det finns en väg mellan Örebro och Karlskoga: **väg Örebro Karlskoga**

Man använder nyckelordet **rutt** (och nyckelordet **framme**) för att lägga in en rutt, till exempel så här för att lägga in en rutt från Örebro till Karlstad: **rutt Örebro Karlskoga Kristinehamn Karlstad framme**

Man använder nyckelordet **slut** för att ange slutet på hela inmatningen.

Här är ett komplett exempel:

```
ort Örebro
ort Karlstad
ort Karlskoga
ort Kristinehamn
ort Arboga
ort Kungsör
ort Oslo
väg Örebro Karlskoga
väg Karlskoga Kristinehamn
väg Kristinehamn Karlstad
rutt Örebro Karlskoga Kristinehamn Karlstad framme
väg Örebro Arboga
väg Arboga Kungsör
rutt Örebro Arboga Kungsör framme
slut
```

Kommandona ska kunna skrivas på fritt format, vilket betyder att blanktecken och radslut inte spelar någon roll, annat än för att skilja orden från varandra.

Vi antar att ortnamn alltid består av ett enda ord.

Uppgift 3: Grammatiker (5 p)

a) (1p) Vilka tokentyper ingår i språket?

b) (4p) Skriv en grammatik för språket. Använd terminalerna från deluppgift a. Om grammatiken inte lämpar sig för implementation i form av en prediktiv recursive-descent-parser, ska du också transformera den så den passar.

Uppgift 4: Träd (3 p)

Ett parse-träd (ibland kallat "konkret syntaxträd") innehåller noder för alla icke-terminaler. I ett syntax-träd (ibland kallat "abstrakt syntaxträd") har man tagit bort alla "onödiga" inre noder, och flyttat upp operatorerna.

Här är ett kortare exempel på inmatning till kartdatabasen:

```
ort Örebro
ort Karlstad
väg Örebro Karlskoga
slut
```

Rita upp ett parse-träd för denna inmatning, med din grammatik från uppgiften ovan.

Uppgift 5: Parsning (6 p)

Skriv en prediktiv recursive-descent-parser för inmatningsspråket. Gör detta i ett språk som åtminstone liknar något vanligt känt programmeringsspråk, till exempel C. 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. Förklara gärna sådant som kan antas vara oklart för läraren.

Du kan anta att det redan finns en funktion som heter **scan**, och att den returnerar typen på nästa token.

Uppgift 6: Några termer (4 p)

Förklara kort vad följande begrepp från kompilatortekniken innebär. Ge gärna exempel.

1. Aktiveringspost ("activation record")
2. Döda data ("dead data")
3. Död kod ("dead code")
4. Referensräkning ("reference counting")
