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

Tentamen i

Kompilatorer och interpretatorer

för Dataingenjörsprogrammet m fl

tisdag 16 augusti 2016

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

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 43. 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 tisdag 6 september 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 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)

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

```
#binclude <stdio.h> // error: invalid preprocessing directive

int main(void) {
    int a;
    ink b; // error: unknown type name 'ink'
    float a; // error: conflicting types for 'a'

    printf("Hej!\n); // error: missing terminating " character
    print("Tal: "); // error: undefined reference to `print'
    scanf("%d", &a);
    printf("Talet = %d\n", a);

    return "OK"; // warning: return makes integer from pointer
}
```

Uppgift 3 (7 p)

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

```
a = 1;
b = 2;
while (a < 3 + b + 4) {
    a = 5;
    b = 6;
    if (a > 7) {
        b = 8;
    }
    else {
        b = 9;
    }
}
a = 10;
```

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 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, med innehåll, när programkörningen kommer till utskriften med printf.

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

int a = 1;

void g(void) {
    int d;
    int *e;
    d = 7;
    e = malloc(sizeof(int));
    *e = 8;
}

int f(int x, int y) {
    int a;
    int *c;
    a = 5;
    x = x + y;
    c = malloc(sizeof(int));
    *c = 6;
    g();
    printf("Här!\n");
    return a;
}

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

Scenario till de övriga uppgifterna

En form av pistolskytte går ut på att man skjuter en serie med fem skott mot en tioringad tavla, och försöker få så många poäng som möjligt. Träffar man med alla skotten i tian, får man femtio poäng:

De svarta klisterlapparna är överklistrade skott från tidigare serier.

För att förenkla poängräkningen vill vi skapa ett program, där man kan mata in poängen med ett särskilt kommandospråk. Här är ett exempel på en fullständig dialog med programmet, där användarens inmatning (med det särskilda kommandospråket) är understruken, och programmets utmatning är raderna som börjar med stjärnor.

```
skytt "Olle"
*** Olle är skytt nummer 1
skytt "Anna Strid"
*** Anna Strid är skytt nummer 2
skytt "Anna Stål"
*** Anna Stål är skytt nummer 3
serie 1 10 10 10 9 9
*** Olle sköt en serie på 48 poäng
serie 2 10 10 10 10 8
*** Anna Strid sköt en serie på 48 poäng
serie 3 10 10 7 7 7
*** Anna Stål sköt en serie på 41 poäng
serie 1 10 10 9 9 8
*** Olle sköt en serie på 46 poäng
serie 2 10 10 10 10 10
*** Anna Strid sköt en serie på 50 poäng
serie 3 10 10 9 9 9
*** Anna Stål sköt en serie på 47 poäng
klar
*** Resultat:
```

```
*** 1. Anna Strid (skytt nummer 2): 98 poäng
*** 2. Olle (skytt nummer 1): 94 poäng
*** 3. Anna Stål (skytt nummer 3): 88 poäng
```

Inmatningen består av rader, där en rad kan börja med "skytt", och ange en skytt som deltar i tävlingen, eller med "serie", och ange en serie som en skytt har skjutit, eller "klar", och ange att inmatningen är slut.

Uppgift 5 (2 p)

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

Uppgift 6 (4 p)

Skriv en grammatik för inmatningsspråket i scenariot. Startsymbolen ska vara **tävling**, som representerar en komplett inmatning enligt scenariot.

Uppgift 7 (3 p)

Rita upp parse-trädet (även kallat konkret syntaxträd) för nedanstående inmatning, enligt grammatiken i uppgiften ovan:

```
skytt "Ragnar"
serie 1 10 10 10 10 10
klart
```

Uppgift 8 (7 p)

Skriv en prediktiv recursive-descent-parser för inmatningsspråket i scenariot, 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**, och att den returnerar typen på nästa token.
