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

Tentamen i

Programmering grundkurs och Programmering C

för D1 m fl, även distanskursen

torsdag 18 augusti 2011

Gäller som tentamen för:
DT1016 Datateknik A, Programmering grundkurs, provkod 0100
DT1007 Datateknik A, Tillämpad datavetenskap, provkod 0410
DT1006 Datateknik A, Programmering C, distans, provkod 0100

Hjälpmedel:	Inga hjälpmedel.
Poängkrav:	Maximal poäng är 40. För godkänt betyg (3 respektive G) krävs 20 poäng.
Resultat och lösningar:	Meddelas via e-post eller på kursens hemsida, http://basen.oru.se/kurser/c/2010-2011-p2/, senast torsdag 8 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!

Prioritet och associativitet hos operatorerna i C

De viktigaste operatorerna:

Prioritet	Kategori	Operator	Associativitet
Högsta	Unära postfixoperatorer	(), [], ->, ., ++, --	vänster
	Unära prefixoperatorer	!, ++, --, +, -, *, &, sizeof, (*typ*)	höger
	Multiplikation mm	*, /, %	vänster
	Addition mm	+, -	vänster
	Jämförelser	<, <=, >=, >	vänster
	Likhetsjämförelser	==, !=	vänster
	Logiskt OCH	&&	vänster
	Logiskt ELLER	||	vänster
Lägsta	Tilldelning	=, +=, -=, *=, /=, %=	höger

Uppgift 1 (1 p)

Vilka värden har följande uttryck?

a) 1 + 2 * 3 + 4

b) 1 - 2 + 3 - 4

c) 1 / 2 + 3 % 4

Uppgift 2 (3 p)

Skriv ett fullständigt C-program (med **#include** och allt) som läser in tre heltal, **start**, **stopp** och **steg**, och sedan skriver ut alla talen mellan **start** och **stopp**, med steget **steg**.

Ett körexempel, med användarens inmatning *kursiverad*:

```
Ange start: 2
Ange stopp: 13
Ange steg: 3
2 5 8 11
```

Om **stopp** är mindre än **start** ska man direkt få ett felmeddelande, och programmet ska avslutas. Exempel:

```
Ange start: 13
Ange stopp: 2
Start måste vara mindre än stopp.
```

I den här och alla andra uppgifter på tentan gäller: Normalt är felhantering en stor del av ett program. Vad ska till exempel hända om användaren skriver **Kalle** när hon egentligen borde mata in ett tal? Här behövs dock ingen felhantering, om så inte särskilt efterfrågas i uppgiften.

Uppgift 3 (2 p)

Vad händer när följande programsnutt körs? Vad skrivs ut?

```
int a = 1, b, c;
float x = 1.0, y, z = 3.0;
c = 3;
while (a < c) {
    printf("x = %f\n", x);
    if (x < z) {
        x = x + 1;
    }
    else {
        a = a + 1;
    }
}
```

Uppgift 4 (1 p)

Skriv en funktion, **safediv**, som tar två flyttal (**x** och **y**) som parameter, och som returnerar kvoten **x**/**y**. Om **y** är noll, ska i stället talet noll returneras.

Uppgift 5 (2 p)

Skriv en **main**-funktion som läser in två flyttal, sen anropar funktionen **safediv** för att räkna ut deras kvot, och till sist skriver ut resultatet.

I den här och alla andra uppgifter på tentan gäller: Om du behöver använda något från en tidigare uppgift eller deluppgift, till exempel anropa en funktion som skrevs i den tidigare uppgiften, så behöver du inte skriva samma kod igen. Du får också göra uppgiften även om du inte gjort den tidigare uppgiften.

Uppgift 6 (3 p)

Skriv en funktion, **how_many_outliers**, som tar fyra argument: en array **a** av flyttal, ett heltal **n** som anger antalet tal i den arrayen, ett flyttal **m** som anger ett nominellt värde, och ett flyttal **p** som anger en procentsats.

Funktionen ska returnera antalet tal i arrayen **a** som avviker med mer än **p** procent från det nominella värdet **m**.

Uppgift 7 (3 p)

Skriv en **main**-funktion som läser in värden på **n**, **a**, **m** och **p**, som sedan anropar **how_many_outliers**, och som till sist skriver ut antalet värden i **a** som avvek mer än **p** procent från **m**.

Det räcker om man kan hantera värden på **n** som är högst 10.

Scenario till de följande uppgifterna

Vi vill lagra data om Sveriges tätorter i datatypen **struct Tatort**:

```
#define NAMN_MAX 20

struct Tatort {
    char namn[NAMN_MAX + 1];
    char kommun[NAMN_MAX + 1];
    int folkmangd;
    double latitud;
    double longitud;
    double elevation;
};
```

Varje tätort har ett namn, som **Örebro** eller **Övre Soppero**. Varje tätort ligger i en viss kommun, till exempel att Övre Soppero ligger i **Kiruna** kommun. (Både ortsnamnet och kommunnamnet kan innehålla flera ord.) Den har också en folkmängd, koordinater i form av latitud och longitud, samt en höjd över havet (elevation).

Vi vet namn och kommun för alla tätorter, men det kan hända att vi inte tagit reda på alla av de övrga uppgifterna. Om någon av dessa har värdet **-1000**, betyder det att uppgiften är okänd.

Uppgift 8 (1 p)

Definiera en variabel av typen **struct Tatort** och initiera den med data om orten **Norra Bro** i **Örebro** kommun, med folkmängden **687**, men där vi ännu inte tagit reda på koordinaterna och höjden över havet.

Uppgift 9 (2 p)

Vi vill kunna visa tätortsposternas innehåll på skärmen. Skriv en funktion som heter **visa_tatort**, och som skriver ut data om en tätort. Funktionen ska ta en tätortspost (eller, om du vill, en pekare till den) som parameter. Värden som är okända (markerade med -1000) ska inte skrivas ut!

Uppgift 10 (2 p)

Skriv en funktion som heter **las_tatort**, och som läser in data om en tätort. Funktionen ska skriva ut lämpliga ledtexter på standardutmatningen, och läsa in data från standardinmatningen (som normalt är kopplad till tangentbordet).

Välj själv vilket av dessa två funktionshuvuden du ska använda:

```
struct Tatort las_tatort()
void las_tatort(struct Tatort *p)
```

Uppgift 11 (2 p)

Skriv en main-funktion som har två lokala variabler av typen **struct Tatort**, och som läser in data om två tätorter till dessa variabler med hjälp av funktionen **las_tatort**. Därefter ska programmet skriva ut data om den tätort som har störst folkmängd, med hjälp av funktionen **visa_tatort**.

Uppgift 12 (2 p)

Jorden är rund, så om man vill räkna ut avståndet mellan två punkter på jordytan som man angett med latitud och longitud, räcker det inte med enkel plangeometri och Pythagoras sats. Men inom Sverige kan man använda den här approximationen:

$$avstånd = \sqrt{(57.03 \cdot (long_1 - long_2))^2 + (111.40 \cdot (lat_1 - lat_2))^2}$$

lat$_1$ och **long$_1$** är den första punktens position, och **lat$_2$** och **long$_2$** den andras. Avståndet anges i kilometer.

Skriv en funktion som heter **avstand**, som tar två tätortsposter som argument, och som returnerar avståndet mellan de två orterna, uträknat med formeln ovan.

Uppgift 13 (4 p)

Skriv en funktion, **max_avstand**, som tar två argument: en array **orter** av orter (typen **struct Tatort**) och ett heltal **n** som anger antalet orter i den arrayen. Funktionen ska returnera avståndet mellan de två orter som ligger längst ifrån varandra. Använd funktionen **avstand**.

Uppgift 14 (6 p)

31 december 2010 fanns det 1956 tätorter i Sverige, med en sammanlagd befolkning av 8015797 invånare, vilket motsvarade 85,1% av hela Sveriges befolkning.

Data om alla dessa tätorter finns på textfilen **tatorter.txt**, med orterna ordnade efter folkmängd. Samma data finns också på binärfilen **tatorter.bin**.

a) (1p) Hitta själv på, och beskriv, ett lämpligt format för textfilen.

b) (4p) Skriv ett C-program som kontrollerar att det verkligen är samma data på de två filerna.

c) (1p) Finns det några principiella problem med att avgöra om det är "samma data"?

Uppgift 15 (6 p)

Skriv ett C-program som lägger till en ort i filen. Den nya orten måste läggas in på rätt plats, enligt sorteringen efter folkmängd. Välj själv om du vill arbeta med textfilen eller binärfilen.

Tips:

- Läs in data om den nya orten med **las_tatort**.
- Det är lättast att kopiera hela den gamla filen till en ny, ort för ort, och att i samband med det lägga till den nya tätorten på rätt plats. Därefter kan man använda den färdiga funktionen **rename** för att byta namn på den nya filen, så den ersätter den gamla:

```
int rename(char *oldpath, char *newpath);
```
