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

Tentamen i

Programmering grundkurs och Programmering C

för D1 m fl, även distanskursen

fredag 15 januari 2010

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 23 poäng. För den som följt campuskursen hösten 2009 ger varje i tid inlämnad inlämningsuppgift med deadline en extra poäng. Den som *inte* gått campuskursen hösten 2009 får dessa (tre) extrapoäng ändå.
Resultat och lösningar:	Meddelas via e-post eller på kursens hemsida, http://www.aass.oru.se/~tpy/c/2009-2010-p2/, senast lördag 6 februari 2010.
Återlämning av tentor:	Efter att resultatet meddelats kan tentorna hämtas på institutionen. Man kan också få sin rättade tenta hemskickad.
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

d) 1 / 2.0 + 3 % 4

Uppgift 2 (1 p)

x, **y** och **z** är flyttalsvariabler av typen **float**. Ange värdet på x, y och z när följande kod har körts.

```
x = 1.0; y = 2.0; z = 3;
if (x < y + z)
    z = z * 2;
while (x < y + 1) {
    x = x * y;
    z = z + 1;
}
```

Uppgift 3 (1 p)

Vi vill kontrollera svaret från uppgift 2 ovan. Skriv därför ett **printf**-anrop som skriver ut de tre variablerna **x**, **y** och **z**.

I den här och alla andra uppgifter på tentan gäller: Om du ska 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 4 (2 p)

Vi jobbar vidare med de tre variablerna **x**, **y** och **z**. **p1** och **p2** är variabler av typen **pekare till float**.

a) Ange värdet på x, y och z när följande kod har körts.

```
p1 = &x;
*p1 = 3.14;
p2 = &y;
*p2 = *p1 * 2;
p1 = &z;
*p1 = *p2;
p1 = p2;
```

b) Vad pekar **p1** respektive **p2** nu på?

Uppgift 5 (2 p)

Skriv en funktion som heter **max3**, som tar tre flyttal av typen **double** som argument, och som returnerar det största av de tre talen.

Uppgift 6 (3 p)

Skriv en funktion som heter **arraymax**, som tar en array med flyttal av typen **double** som argument, tillsammans med ett heltalsargument som anger antalet tal i arrayen, och returnerar det största av talen i arrayen. (Vi behöver inte bry oss om fallet att arrayen innehåller noll tal.)

Uppgift 7 (3 p)

Vi vill provköra funktionerna **max3** och **arraymax** från uppgifterna ovan. Skriv därför en **main**-funktion, som anropar dem med lämpliga argument, och skriver ut resultatet.

Vi ska bara anropa varje funktion en gång, så det finns alltså bara ett enda testfall per funktion. Försök göra dessa två testfall så bra som möjligt!

Uppgift 8 (1 p)

Fig. 1. Ett stort monster.

Fig. 2. Ett litet monster.

Vi ska göra ett spel med monster i. Uppgifter om monstren ska lagras i poster ("structar") av typen **struct Monster**. En monsterpost innehåller monstrets **namn**. Namnet är ett enda ord utan mellanslag, och kan innehålla upp till tio tecken, som i **Stor-Fulis**, men inte **Makro-Fulis**. Monstret har också en **tuffhet**, som är ett flyttal, och ett **antal tänder**, som är ett heltal.

Definiera posttypen **struct Monster**.

Uppgift 9 (1 p)

Definiera en variabel av typen **struct Monster** och initiera den med data om monstret **Uschlo**, som har en tuffhet på **12.5** och **300** tänder.

Uppgift 10 (2 p)

Vi vill kunna visa monsterposternas innehåll på skärmen. Skriv en funktion som heter **visa_monster**, som skriver ut data om ett monster. Funktionen ska ta en monsterpost (eller, om du vill, en pekare till den) som parameter. Så här kan utskriften se ut:

```
Namn: Uschlo
Tuffhet: 12.50
Tänder: 300
```

Uppgift 11 (2 p)

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

Du får själv välja om du vill att funktionshuvudet ska se ut så här:

```
struct Monster las_monster()
```

eller så här:

```
void las_monster(struct Monster *p)
```

Uppgift 12 (1 p)

Om ett monsters tuffhet går ner under noll, har monstret dött. Skriv därför en funktion som heter **lever**, och som tar en monsterpost som argument (eller, om du vill, en pekare till den). Funktionen ska returnera ett sant värde om monstret lever, och ett falskt värde om det är dött.

Uppgift 13 (3 p)

Monstren ska kunna slåss mot varandra. Skriv därför en funktion som heter **attackera**, och som låter ett monster attackera ett annat. Reglerna är enkla: Det attackerande monstret försöker bita sitt stackars offer. Det har 50 procents chans att träffa med sitt bett. Om det träffar med bettet, förlorar offret lika mycket tuffhet som antalet tänder som bitaren har.

Exempel:

Monstret **Varvulven** har **36** tänder och tuffheten **99.2** Monstret **Hajen** har **100** tänder och tuffheten **30.5**. Varulven attackerar Hajen, och träffar. Efter attacken har Hajen tuffheten **-5.5**. Det betyder också att Hajen dog, eftersom dess tuffhet nu är mindre än noll.

Tips: Funktionen **rand** returnerar ett slumpmässigt heltal.

Funktionen **attackera** ska ta två monsterposter (eller pekare till dem) som argument: det attackerande monstret, och det attackerade monstret.

Uppgift 14 (4 p)

Skriv en main-funktion som har två lokala variabler av typen **struct Monster**, och som läser in data om två monster till dessa variabler med hjälp av funktionen **las_monster**. Därefter ska monstren försöka attackera varandra med hjälp av två anrop till funktionen **attackera**. Avslutningsvis ska programmet använda funktionen **visa_monster** för att skriva ut data om de monster som fortfarande lever. (Använd funktionen **lever** för att kolla detta.)

Kom ihåg: Om du ska 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 15 (4 p)

Vi vill skapa en fil med monster på. Skriv ett C-program som upprepat läser in monster med hjälp av funktionen **las_monster**, och skriver alla monsterdata på en fil. Välj själv om det ska vara en text- eller en binärfil. Välj också själv hur inmatningen ska avslutas.

Om filen inte går att öppna, ska programmet skriva ut ett felmeddelande om detta, och sen avslutas.

Uppgift 16 (9 p)

Skriv ett C-program som läser filen från uppgiften ovan, och låter monstren attackera varandra, tills endast ett monster finns i livet.

Börja med att läsa in monstren till en array av monsterposter. Det ska få plats högst 1000 monster i arrayen. Om det finns fler monster på filen, ska de ignoreras, men programmet ska också skriva ut en varning om detta.

Om filen inte går att öppna, ska programmet skriva ut ett felmeddelande om detta, och sen avslutas.

Därefter ska monstren attackera varandra, och detta ska fortsätta tills det bara finns ett enda levande monster kvar. Avslutningsvis ska det levande monstrets data skrivas ut.

Du får själv välja hur man bestämmer vilka monster som attackerar, och hur de väljer sina offer. Man kan slumpa fram det, eller alltid låta det första levande monstret i arrayen attackera det andra, eller hur du vill.

Programmet ska använda sig av funktionerna **attackera**, **lever** och **visa_monster**.
