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

Tentamen i distanskursen

Programmering C

torsdag 18 augusti 2016

Gäller som tentamen för:
DT104G Programmering C, provkod 0100
DT1006 Datateknik A, Programmering C, distans, provkod 0100

Campuskursen "Programmering grundkurs" har en egen tenta.

Hjälpmedel:	Ordbok för översättning.
Poängkrav:	Maximal poäng är 44. För godkänt betyg (3 respektive G) krävs 22 poäng.
Resultat och lösningar:	Meddelas via e-post senast torsdag 8 september 2016.
Återlämning av tentor:	Efter att resultatet meddelats kan tentorna hämtas 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!

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

Några användbara biblioteksfunktioner

stdlib.h

```
int rand(void);
void srand(unsigned int seed);
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
void exit(int status);
void qsort(void *base, size_t nmemb, size_t size,
           int(*compar)(const void *, const void *));
```

stdio.h

```
FILE *fopen(const char *path, const char *mode);
int fclose(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int ungetc(int c, FILE *stream);
char *fgets(char *s, int size, FILE *stream);
char *gets(char *s);
int putc(int c, FILE *stream);
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...);
int snprintf(char *str, size_t size, const char *format, ...);
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

string.h

```
size_t strlen(const char *s);
char *strcpy(char *dest, const char *src);
char *strncpy(char *dest, const char *src, size_t n);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
char *strcat(char *dest, const char *src);
char *strncat(char *dest, const char *src, size_t n);
char *strstr(const char *haystack, const char *needle);
void *memmove(void *dest, const void *src, size_t n);
```

ctype.h

```
int isalnum(int c);
int isalpha(int c);
int isblank(int c);
int isdigit(int c);
int islower(int c);
int isprint(int c);
int ispunct(int c);
int isspace(int c);
int isupper(int c);
```

math.h

```
double sqrt(double x);
double pow(double x, double y);
```

Uppgift 1 (1 p)

Vilka värden har följande uttryck?

a) `1 - 2 - 3`

b) `4 + 4 % 3`

c) `1+2 * 3`

d) `7 / 2 * 2`

Uppgift 2 (2 p)

a, **b** och **i** är heltalsvariabler. Ange värdet på variablerna då följande kod körts.

```
a = 1;
b = 2;
i = 3;
while (i < 10) {
    a = b;
    if (i < 4 || i == 8)
        i += 3;
    else
        i++;
    b = b + 1;
}
```

Uppgift 3 (3 p)

Skriv ett komplett C-program (med **#include** och allt) som läser in tre heltal, vi kan kalla dem **a**, **b** och **c**, och sen skriver ut alla tal från **a** till **b** (inklusive de talen) utom talet **c** (om det nu ligger mellan **a** och **b**),

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 4 (4 p)

Skriv ett C-program som läser reella tal (som lagras internt som flyttal) från standardinmatningen. Programmet ska avslutas när talet 0 matas in, och då skriva ut medelvärdet av de inmatade talen *utom* det högsta och det lägsta (och den avslutande nollan). För enkelhets skull kan vi anta att alla inmatade tal kommer att vara olika, och att inmatningen består av minst tre tal.

Scenario till de följande uppgifterna

Vi ska göra ett spel där man går runt i en grotta och, som omväxling, *inte* slåss med monster, utan matar monstren med glass och godis.

Ett monster har ett unikt nummer, ett (inte nödvändigtvis) unikt namn och en storlek. När man matar monstret växer det, dvs storleken ökar. Storleken är ett flyttal, till exempel **10.2**, och namnet kan innehålla högst tio tecken, till exempel **Söt-Wumpus**.

Varje rum i grottan har ett unikt nummer, och det innehåller upp till fem monster.

Uppgift 5 (5 p)

a)

Skapa en posttyp som heter **struct Monster**, och som innehåller data om ett monster.

b)

Definiera en variabel av typen **struct Monster** och initiera den med data om monster nummer **17**, som heter **Gluggo** och har storleken **2000**.

c)

Skapa en posttyp som heter **struct Rum**, och som innehåller data om ett rum, inklusive en array med de högst fem monster som finns i rummet. Det är praktiskt om posttypen innehåller en variabel som anger antalet monster som för tillfället finns i arrayen.

d)

Spelet kan hantera högst 100 rum, så skapa en arrayvariabel som heter **rum** och som har plats för 100 rumsposter. Skapa också en variabel som heter **antal_rum**, för att hålla reda på hur många platser i arrayen som för tillfället används.

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 6 (4 p)

a)

Skriv funktionen **totalt_antal_monster**. Man ska kunna anropa den med arrayen **rum** och antalet rum som parametrar, och så ska den returnera det totala antalet monster som finns i alla rummen.

b)

Visa de kodrader som behövs för att anropa funktionen **totalt_antal_monster**, och därefter skriva ut resultatet.

Uppgift 7 (2 p)

Vi vill kunna visa monsterposternas innehåll på skärmen. Skriv en funktion som heter **visa_monster**, som skriver ut uppgifterna om ett monster. Funktionen ska ta monsterposten som parameter. Exempel på hur en utskrift skulle kunna se ut:

```
Nummer: 17
Namn: Gluggo
Storlek: 2000.000000
```

Uppgift 8 (3 p)

Vi vill också kunna visa rumsposterna innehåll på skärmen. Skriv en funktion som heter **visa_rum**, och som skriver ut uppgifterna om ett rum. Funktionen ska ta rumsposten som parameter. Använd funktionen **visa_monster** för att skriva ut data om de monster som finns i rummet. Exempel på hur en utskrift skulle kunna se ut:

```
Rumsnummer: 19
Rummet innehåller 2 monster:
Nummer: 17
Namn: Gluggo
Storlek: 2000.000000
Nummer: 18
Namn: Miniglugg
Storlek: 17.230000
```

Uppgift 9 (3 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 hur funktionshuvudet ska se ut. Här är två förslag:

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

Uppgift 10 (3 p)

Skriv en funktion som heter **las_rum**, och som läser in data om ett rum, inklusive de monster som finns i rummet. Funktionen ska skriva ut lämpliga ledtexter på standardutmatningen, och läsa in data från standardinmatningen (som normalt är kopplad till tangentbordet). Den ska anropa funktionen **las_monster** för att läsa in data om monster.

Du får själv välja hur funktionshuvudet ska se ut. Här är två förslag:

```
struct Rum las_rum(void)
void las_rum(struct Rum *p)
```

Uppgift 11 (4 p)

Skriv en **main**-funktion som har två lokala variabler av rumstypen från uppgifterna ovan, och som läser in data om två rum till dessa variabler med hjälp av funktionen **las_rum**. Avslutningsvis ska den använda funktionen **visa_monster** för att skriva ut det största av de monster som finns i de två rummen.

Uppgift 12 (10 p)

a)

Skriv funktionen **spara_rum**, som sparar innehållet i en rumspost på en fil. Funktionen ska ha två parametrar: rumsposten som ska sparas, och namnet på den fil som ska skapas. Om filen inte gick att öppna, ska ett felmeddelande skrivas ut, och programmet ska avslutas.

b)

Skriv funktionen **hemta_rum**, som läser in innehållet i en rumspost från en fil. Funktionen ska ta filnamnet som parameter, men du får själv välja hur rumsposten ska returneras. Om inläsningen misslyckas, antingen för att filen inte gick att öppna eller för att det uppstod ett fel vid inläsningen, ska ett felmeddelande skrivas ut, och programmet ska avslutas.

c)

Skriv en **main**-funktion som har två lokala variabler av rumstypen, kallade **rum1** och **rum2**. Den ska läsa in data om ett rum till **rum1** med hjälp av funktionen **las_rum**, och därefter spara det rummets data på en fil med hjälp av funktionen **spara_rum**. Därefter ska den använda funktionen **hemta_rum** för att läsa in rumsuppgifterna till variabeln **rum2**, och slutligen visa dem med funktionen **visa_rum**.
