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

Tentamen i distanskursen

Programmering C

torsdag 17 augusti 2017

Gäller som tentamen för:
DT104G Programmering C, provkod 0100

Hjälpmedel:	Ordbok för översättning.
Poängkrav:	Maximal poäng är 37. För godkänt betyg (3 respektive G) krävs 18 poäng.
Resultat och lösningar:	Meddelas via e-post senast torsdag 7 september 2017.
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);
```

För uppgifterna på tentan gäller: Om du ska använda något från en tidigare uppgift eller deluppgift, till exempel anropa en funktion från den tidigare uppgiften, behöver du inte skriva koden på nytt. Du får också göra uppgiften även om du inte gjort den tidigare uppgiften som den bygger på.

För uppgifterna 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 1 (1 p)

Vilka värden har följande C-uttryck?

a) `1 + 2 * 3`

b) `1 < 2 || 3 != 4`

c) `2 + 4 / 3`

d) `1 - 2*3 - 4*5`

Uppgift 2 (7 p)

Hur många stjärnor skrivs ut av var och en av följande loopar?

a)

```
for (int i = 0; i < 10; ++i)
    printf("*");
```

b)

```
for (int i = 0; i <= 10; ++i)
    printf("*");
```

c)

```
int i;
i = 1;
while (i <= 10) {
    printf("*");
    i++;
}
```

d)

```
for (int i = 0; i < 10; ++i) {
    printf("*");
    i = i + 2;
}
```

e)

```
i = 10;
while (i > 0) {
    printf("*");
    i = i - 1;
}
```

f)

```
i = 1;
do {
    printf("*");
} while (++i < 10);
```

g)

```
for (int i = 10; i < 10; i++) {
    printf("*");
}
```

Scenario till de följande uppgifterna

En **talserie** består av ett antal heltal i en viss följd. För enkelhets skull antar vi att alla talserier innehåller högst tio tal.

Här är några definitioner som vi skapat för att representera talserier i C:

```
#define MAX_ANTAL_TAL 10

struct Talserie {
    int antal;
    int talen[MAX_ANTAL_TAL];
};

typedef struct Talserie Talserie;
```

Uppgift 3 (10 p)

a) (1p)

Definiera en variabel som har datatypen **struct Talserie**, och initiera den med data om talserien **3, 3, 9, 8, 3**.

b) (2p)

Vi vill kunna visa data om en talserie på skärmen. Skriv därför funktionen **visa_talserie**, som tar en talserie (eller en pekare till en talserie, om du hellre vill det) som parameter, och skriver ut alla talen i talserien på standardutmatningen.

c) (2p)

Skriv funktionen **lika_talserier**, som tar två talserier (eller pekare till talserier, om du hellre vill det) som parametrar, och returnerar ett sant värde om de två talserierna är likadana, dvs innehåller samma tal, i samma ordning. Om serierna inte är lika ska ett falskt värde returneras.

d) (3p)

Skriv funktionen **las_talserie**, som läser in en talserie. Funktionen ska skriva ut lämpliga ledtexter på standardutmatningen, läsa in talen från standardinmatningen (som normalt är kopplad till tangentbordet), och på lämpligt sätt returnera den inlästa talseriens data till den anropande funktionen.

e) (2p)

Skriv en **main**-funktion som har två lokala variabler av typen **struct Talserie**, och som läser in data om två talserier till de variablerna med hjälp av funktionen **las_talserie**. Därefter ska programmet använda funktionen **lika_talserier** för att avgöra ifall serierna är likadana, och skriva ut resultatet av jämförelsen.

Uppgift 4 (12 p)

I uppgiften ovan lagrades talserier i en array med fast storlek, och vi kunde inte lagra serier med fler än 10 tal. Vi kan förstås ändra makrot **MAX_ANTAL_TAL** så det finns plats för fler tal, men om vi sätter det till, exempelvis, **1000000** slösar vi bort ganska mycket plats om en serie är kort, och om en serie är 1000001 tal lång så får den i alla fall inte plats.

Gör därför om hela uppgiften, men nu ska talen lagras i en länkad lista i stället!

Förutom alla deluppgifterna a-f måste du först skriva de typdefinitioner som behövs, dvs de som var givna i scenariot för versionen med arrayer.

Uppgift 5 (7 p)

a) Skriv funktionen **skriv_till_fil**, som skriver en talserie till en fil. Funktionen ska ha två parametrar: en talserie (eller en pekare till en talserie, om du hellre vill det) och ett filnamn i form av en textsträng. En enda serie ska lagras på varje fil.

b) Skriv funktionen **las_fran_fil**, som läser en talserie från en fil. Funktionen ska ta ett filnamn i form av en textsträng, och på lämpligt sätt returnera den inlästa serien till den anropande funktionen.

c) Skriv en **main**-funktion som har två lokala variabler, **serie1** och **serie2**, av typen **struct Talserie**, och som skriver serien i **serie1** till en fil med hjälp av funktionen **skriv_till_fil**, och därefter läser in serien från filen till **serie2** med hjälp av funktionen **las_fran_fil**.

I deluppgift a och b gäller att om filen inte går att öppna, ska ett felmeddelande skrivas ut, och programmet ska avslutas.
