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

Tentamen i distanskursen

Programmering C

lördag 3 juni 2017

Gäller som tentamen för:
DT104G Programmering C, provkod 0100

Hjälpmedel:	Ordbok för översättning.
Poängkrav:	Maximal poäng är 40. För godkänt betyg (3 respektive G) krävs 20 poäng.
Resultat och lösningar:	Meddelas via e-post senast lördag 24 juni 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 *str
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FI
```

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) `2 + 2 * 2`

b) `2 / 5`

c) `2.0 / 5.0`

d) `2+3 * 2+3`

Uppgift 2 (3 p)

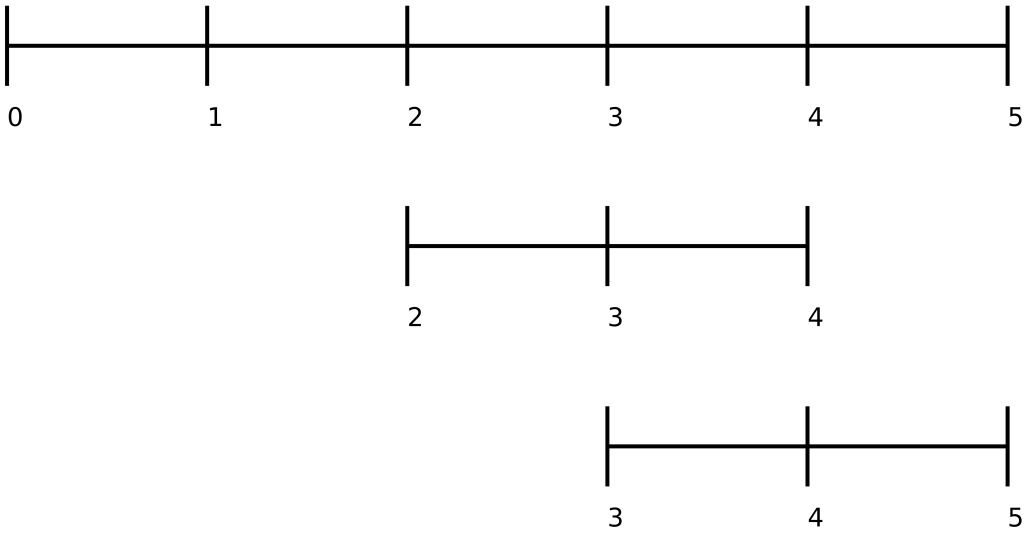
Vad skrivs ut när vi kör följande C-program?

```
#include <stdio.h>

int f(int x, int y) {
    int z;
    z = 0;
    for (int i = 0; i < x; ++i)
        ++z;
    for (int i = 0; i < y; ++i)
        z++;
    return z;
}

int main(void) {
    int x, y, z;

    x = 1;
    y = 2;
    z = x + y;
    printf("z (1) = %d\n", z);
    x = 3;
    y = 4;
    printf("z (2) = %d\n", z);
    printf("f(x, y) = %d\n", f(x, y));
    printf("z (3) = %d\n", z);

    return 0;
}
```

Om någon variabel har ett odefinierat värde ("skräp") när den ska skrivas ut, ange detta!

Scenario till de följande uppgifterna

Ett **intervall** består av en sammanhängande följd av heltal, och anges med en startpunkt och en slutpunkt. Här har vi ritat upp tre intervall:



Det första intervallet består av de sex talen 0-5. Det första intervallet består av de tre talen 2-4. Det tredje intervallet består av de tre talen 3-5.

För enkelhets skull antar vi att alla intervall innehåller minst ett tal.

Uppgift 3 (19 p)

a) (1p)

Skapa datatypen **struct Intervall**, som ska användas för att representera ett intervall enligt ovan.

b) (1p)

Definiera en variabel som har datatypen **struct Intervall**, och initiera den med data om intervallet 0-5.

c) (2p)

Vi vill kunna visa data om intervall på skärmen. Skriv därför funktionen **visa_intervall**, som tar ett intervall (eller en pekare till ett intervall, om du hellre vill det) som parameter, och skriver ut intervallets start- och slutpunkt, med lämpliga ledtexter, på standardutmatningen.

d) (2p)

Vi vill kunna skriva ut alla talen i ett intervall på skärmen. Skriv därför funktionen **visa_tal**, som tar ett intervall (eller en pekare till ett intervall, om du hellre vill det) som parameter, och skriver ut alla talen i intervallet. Exempel på hur det skulle kunna se ut för intervallet 0-5:

```
[0 1 2 3 4 5]
```

e) (1p)

Skriv funktionen **langd** som tar ett intervall (eller en pekare till ett intervall, om du hellre vill det) som parameter, och returnerar intervallets längd. Exempelvis har intervallet 3-5 längden 2.

f) (1p)

Skriv funktionen **antal_tal** som tar ett intervall (eller en pekare till ett intervall, om du hellre vill det) som parameter, och returnerar antalet tal i intervallet. Exempelvis finns det 3 tal i intervallet 3-5.

g) (2p)

Ett intervall är en *delmängd* av ett annat intervall om alla tal i det första intervallet också finns i det andra intervallet. Exempelvis är både intervallet 2-4 och intervallet 3-5 delmängder av intervallet 0-5. 2-4 är inte en delmängd av 3-5, och 0-5 är inte en delmängd av 2-4 eller 3-5. Ett intervall är alltid en delmängd av sig självt. Skriv funktionen **delmangd** som tar två intervall (eller pekare till intervall, om du hellre vill det) som parametrar, och returnerar ett sant värde om det första intervallet är en delmängd av det andra. Annars ska ett falskt värde returneras.

h) (3p)

Två intervall är *överlappande* om det finns minst ett tal som ingår i båda intervallen. Exempelvis är intervallen 2-4 och 3-5 överlappande, och intervallen 0-5 och 2-4 är överlappande. Intervallen 0-5 och 5-6 är överlappande, men intervallen 0-5 och 6-7 är inte överlappande. Skriv funktionen **overlappande** som tar två intervall (eller pekare till intervall, om du hellre vill det) som parametrar, och returnerar ett sant värde om intervallen är överlappande. Annars ska ett falskt värde returneras. Ett tips: Man kan ange intervallen i godtycklig ordning, så både ett anrop med 2-4 och 3-5, och ett anrop med 3-5 och 2-4, ska returnera ett sant värde.

i) (4p)

Skriv funktionen **las_intervall**, som läser in ett intervall. Funktionen ska skriva ut lämpliga ledtexter på standardutmatningen, läsa in intervallets start- och slutpunkt från standardinmatningen (som normalt är kopplad till tangentbordet), och på lämpligt sätt returnera det inlästa intervallets data till den anropande funktionen.

Funktionen ska ha felkontroll, och ska hantera problem som att man matar in ett intervall där startpunkten är större än slutpunkten, skriver **Kalle** i stället för ett tal, och så vidare.

j) (2p)

Skriv en **main**-funktion som har två lokala variabler av typen **struct Intervall**, och som läser in data om två intervall till de variablerna med hjälp av funktionen **las_intervall**. Därefter ska programmet använda funktionen **antal_tal** för att avgöra vilket intervall som innehåller flest tal, och skriva ut detta intervall med hjälp av funktionen **visa_intervall**.

Uppgift 4 (6 p)

Vi ska skapa en länkad lista bestående av intervall.

Skriv ett C-program som först frågar efter och läser in hur många intervall som ska hanteras. Därefter ska programmet läsa in så många intervall, med hjälp av funktionen **las_intervall**, och lagra dem i en länkad lista.

Därefter ska programmet gå igenom den länkade listan, hitta det intervall som innehåller flest tal, med hjälp av funktionen **antal_tal**, och slutligen skriva ut detta intervall med hjälp av funktionen **visa_intervall**.

Uppgift 5 (5 p)

Skriv ett C-program som först frågar efter och läser in hur många intervall som ska hanteras. Därefter ska programmet läsa in så många intervall, med hjälp av funktionen **las_intervall**, och spara dem på en fil.

Om filen inte går att öppna, ska ett felmeddelande skrivas ut, och programmet ska avslutas.

Uppgift 6 (5 p)

Skriv ett program som läser filen från uppgiften ovan, hittar det intervall som innehåller flest tal, med hjälp av funktionen **antal_tal**, och slutligen skriver ut detta intervall med hjälp av funktionen **visa_intervall**.

Om filen inte går att öppna, ska ett felmeddelande skrivas ut, och programmet ska avslutas.

Uppgift 7 (1 p)

Gör en uppskattning av hur många intervall som kan hanteras, på din vanliga dator, med den länkade listan i uppgift 4 respektive med filen i uppgift 5 och 6. Visa hur du tänkt och räknat!
