Örebro universitet
Institutionen 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 20 augusti 2015

Gäller som tentamen för:
DT1029 Datateknik A, Programmering grundkurs, provkod 0100
DT1030 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 senast torsdag 10 september 2015.
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

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 (3 p)

Vad skriver det här C-programmet ut?

```
#include <stdio.h>

int main(void) {
    int a, b, c;
    float x, y, z, t;

    a = 0;
    for (b = 0; b < 10; ++b) {
        ++a;
    }
    for (c = 0; c < 100; ++c) {
        a++;
    }
    printf("a = %d\n", a);
    x = 1.0; y = 2.0; z = 3.0;
    x = y + z;
    for (b = 0; b < 3; ++b) {
        z = z + 1;
        printf("b = %d, x = %f\n", b, x);
    }
    printf("x = %f\n", x);
    printf("summan = %f\n", x + y);
    printf("uttrycket = %f\n", x+1 * y);
    printf("a = %d\n", a);
    printf("t = %f\n", t);

    return 0;
}
```

En del variabler kan ha ett odefinierat värde ("skräp") när de ska skrivas ut. Ange då detta!

Uppgift 2 (4 p)

Vi vill kunna beräkna följande uttryck:

$$\sqrt{x - \frac{1}{R\sqrt{x^2 - y^3}}}$$

Skriv därför ett komplett C-program som först läser in värdena på x, y och R, beräknar uttrycket, och till sist skriver ut uttryckets värde. Beräkningarna ska ske med flyttal.

Om nämnaren i divisionen (dvs deluttrycket under divisionsstrecket) är noll, eller om deluttrycket inuti något av rottecknen är mindre än noll, går uttrycket inte att beräkna. I så fall ska programmet inte försöka beräkna uttrycket, utan det ska i stället skriva ut ett informativt och rättvisande felmeddelande om saken.

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.

I den här och alla andra uppgifter på tentan gäller:
Man kan strunta i detaljer som bara behövs just när man utvecklar konsolprogram i Visual Studio, som konstiga teckenkoder för ÅÄÖ, eller att fönstret med programkörningen försvinner när programmet avslutas.

Uppgift 3 (5 p)

Skriv en C-funktion som skriver ut en array av heltal. Funktionen ska ta tre argument: en array av heltal, ett heltal som anger antalet element i arrayen, och en öppen textfil (av typen **FILE ***) som heltalen ska skrivas på.

Funktionen ska inte returnera något värde.

Talen i arrayen ska skrivas med ett kommatecken och ett mellanslag mellan varje tal. Det ska finnas exakt ett blanktecken före det första talet och efter det sista talet. Runt arrayen ska hakparenteser (dvs **[** och **]**) skrivas. Utmatningen ska avslutas med ett radslutstecken (**\n**). Godtyckligt stora arrayer ska kunna hanteras.

Exempel på utskrifter:

```
[ 1, 2, 9, 10, 0, 33, -6, 2 ]

[ 0, 0, 0, 0 ]

[ ]
```

Uppgift 4 (4 p)

Skriv en main-funktion som först frågar efter antalet heltal som ska hanteras, sedan läser in de talen i en array, därefter frågar efter namnet på en fil, och till sist skriver dem på den filen med hjälp av funktionen från uppgiften ovan. Högst 100 tal behöver kunna hanteras.

I den här och alla andra uppgifter på tentan gäller:
Om du behöver använda något från en tidigare uppgift eller deluppgift, till exempel utnyttja en datatyp eller 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.

I den här och alla andra uppgifter på tentan gäller:
Normalt ska man aldrig använda funktionen **gets**, utan i stället till exempel **fgets**. Här kan du dock använda **gets**.

Uppgift 5 (5 p)

Skriv en C-funktion som tar en sträng som argument, och returnerar antalet av den siffra ('0'..'9') som förekommer flest gånger i strängen. Om funktionen anropas med strängen "kalle" som argument, returnerar den alltså 0. Om den anropas med "kalle7" som argument, returnerar den 1. Om den anropas med "xx447xxx4" som argument, returnerar den 3.

Uppgift 6 (3 p)

Vi vill testa funktionen i uppgiften ovan. Skriv därför en main-funktion som anropar funktionen med tre lämpliga testfall, och skriver ut ett meddelande om den ger ett annat svar än det väntade.

Uppgift 7 (16 p)

Du har just fått jobb som programmerare på Svenska Flipperspelsfabriken AB, som bygger klassiska flipperspel:

I Svenska Flipperspelsfabrikens senaste och mest avancerade flipperspel sitter en liten dator, som styr spelet och som ska programmeras i C. Man har redan skrivit hela programmet, utom administrationen av resultatlistan ("high-score-listan"). Du har nu fått i uppgift att skriva den delen av programmet.

Listan ska hålla reda på de tio spelare som fått flest poäng. Förutom hur många poäng de fått, ska deras namn, i form av tre bokstäver, lagras. Exempel på hur en lista kan se ut:

```
Placering   Namn    Poäng
---------   ----    -----

    1       TPM   9182300
    2       TPM   8819690
    3       KSD   5810200
    4       TPM   5628000
    5       TJO   3671110
    6       TAO   1810990
    7       KSD    757890
    8       TPM    707620
    9       TAO     32450
   10       OWL        80
```

Listans data ska lagras i poster ("struct") i primärminnet (alltså inte på en fil), men i övrigt får du själv avgöra hur datastrukturen ska se ut. Du behöver aldrig lagra mer än tio poster.

a) (2p) Definiera den eller de variabler som behövs. Ge dem också lämpliga initialvärden i definitionerna. Glöm inte att definiera datatyperna, om du använder några.

b) (2p) Skriv en funktion **skriv_listan**, som skriver ut listan. (Det här flipperspelet är ovanligt avancerat, så du kan använda vanliga printf-satser.) Det behöver inte se ut exakt som i exemplet.

c) (2p) Skriv en funktion **ska_lagras**, som tar ett poängtal som argument, och som returnerar ett sant värde om detta poängtal är tillräckligt högt för att få plats i listan. I annat fall ska funktionen returnera ett falskt värde. Tänk på att de tio första resultaten alltid ska lagras.

d) (5p) Skriv en funktion **registrera_resultat**, som tar två argument: ett namn (en sträng på tre bokstäver) och ett poängtal (ett heltal). Poängtalet är alltid större än eller lika med noll. Funktionen ska stoppa in detta resultat på lämplig plats i resultatlistan. Om poängtalet var för lågt, så att resultatet inte ska in i listan, ska listan inte ändras. Tänk på att de tio första resultaten, innan listan är full, alltid ska lagras.

e) (2p) Flipperspelet innehåller även en hårddisk. Skriv funktionen **spara_resultat**, som skriver resultatlistan på en fil.

f) (2p) Skriv funktionen **hemta_resultat**, som läser in resultatlistan igen från filen.

g) (1p) Varför ska funktionen heta **hemta_resultat**, med ett understreck mellan orden, och inte **hemta-resultat**, med ett minustecken (som även används som bindestreck)?
