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

Tentamen i distanskursen

Programmering C

onsdag 13 januari 2016

Gäller som tentamen för:
DT104G Programmering C, provkod 0100
DT1006 Datateknik A, Programmering C, distans, provkod 0100
DT1029 Datateknik A, Programmering grundkurs, provkod 0100
DT1030 Datateknik A, Tillämpad datavetenskap, provkod 0410

Obs! Inte DT106G Programmering grundkurs, provkod 0100. Den har en egen tenta!

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 onsdag 3 februari 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 (2 p)

Vad skrivs ut när vi kör följande C-program?

```
#include <stdio.h>

int main(void) {
    int a, b = 3;
    float x, y;

    a = 1;
    while (a < b) {
        x = a / b;
        y = y + 1;
        a = a + 2;
        b = b + 1;
        printf("Hej!\n");
    }
    printf("a = %d, b = %d, x = %f, y = %f\n", a, b, x, y);

    return 0;
}
```

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 2 (4 p)

Vi vill kunna beräkna följande uttryck:

$$\frac{1}{\sqrt{x^2-y^2}}\left(\frac{\sqrt{x^2-y^2}}{xy+1}-\sqrt{x^2-y^2}\right)^3$$

Skriv därför ett komplett C-program (med **#include** och allt) som först läser in värden på x och y, beräknar uttrycket, och till sist skriver ut uttryckets värde. Beräkningarna ska ske med flyttal.

Om deluttrycket **xy + 1** är lika med noll, eller om deluttrycket under roten är mindre än eller lika med 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.

Uppgift 3 (3 p)

Skriv ett komplett C-program (med **#include** och allt) som läser in ett tvåsiffrigt heltal, och sen ritar upp en fyrkant på skärmen med hjälp av asterisker ("*"). Höjden av fyrkanten ska anges av tiotalssiffran i det inmatade talet, och bredden av entalssiffran.

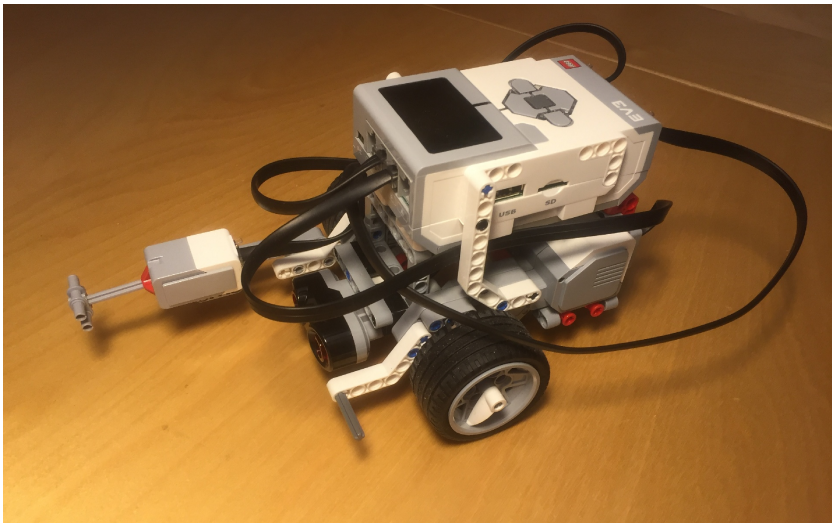
Exempel på hur en programkörning kan se ut, med användarens inmatning i **fetstil**:

```
Ange ett tvåsiffrigt heltal: 74

****
****
****
****
****
****
****
```

Scenario

Här är en liten robot:



Vi ska göra ett program för att simulera sådana robotar, som åker omkring på en spelplan med x- och y-koordinater.

Exempel på hur en programkörning kan se ut, med användarens inmatning i **fetstil**:

```
Hur många robotar finns det? 3
Ange robotens nummer: 1
Ange robotens namn: Olle
Ange robotens x-koordinat: 3
Ange robotens y-koordinat: 4
Ange robotens nummer: 2
Ange robotens namn: Robotron
Ange robotens x-koordinat: 3
Ange robotens y-koordinat: 4
Ange robotens nummer: 3
Ange robotens namn: Olle
Ange robotens x-koordinat: 3
Ange robotens y-koordinat: 3
Både robot 1 och robot 3 heter 'Olle'.
2 robotar på position x = 3, y = 4
```

Som vi ser ska användaren först mata in hur många robotar det finns. Sen ska användaren mata in data om robotarna. Varje robot har ett nummer (i form av ett heltal), ett namn (som är en textsträng som består av ett enda ord), och en position bestående av en x- och en y-koordinat (båda heltal).

Därefter ska programmet kontrollera om det finns flera robotar som har samma namn, och det ska skriva ut nummer och namn för alla par av robotar med samma namn.

Programmet ska också kontrollera om flera robotar finns på samma position på spelplanen. För varje sådan position ska det skriva ut koordinaterna och antalet robotar på den positionen.

Uppgift 4 (10 p)

I den här uppgiften ska vi förbereda oss för att skriva robotprogrammet i scenariot genom att skapa och provköra en datatyp som lagrar data om robotar.

> 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.

a) (1p) Skapa fyra preprocessormakron med **#define**: **MAX_NAMN** som anger längsta tillåtna namnet på en robot, **MAX_ROBOTAR** som anger det högsta antal robotar som vi kan hantera, samt **MAX_X** och **MAX_Y**, som anger de högsta tillåtna x- och y-koordinaterna. Välj rimliga värden.

b) (2p) Skapa posttypen **struct Robot**, som innehåller data om en robot, med nummer, namn och koordinater.

c) (2p) Vi vill kunna visa robot-posternas innehåll på skärmen. Skriv en funktion som heter **visa_robot**, och som skriver ut data om en robot. Funktionen ska ta en robotpost (eller, om du vill, en pekare till den) som parameter. Här är ett exempel på hur utskriften kan se ut:

```
Nummer: 1
Namn: Olle
Position: x = 3, y = 4
```

d) (3p) Skriv en funktion som heter **las_robot**, och som läser in data om en robot. Funktionen ska skriva ut lämpliga ledtexter på standardutmatningen, och läsa in data från standardinmatningen (som normalt är kopplad till tangentbordet). Den inlästa posten ska returneras som returvärde från funktionen, eller via en pekare.

> 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**. (Därmed inte sagt att någon av dessa funktioner nödvändigtvis är det bästa valet i den här uppgiften.)

e) (2p) Vi vill provköra funktionerna ovan. Skriv en **main**-funktion som har två lokala variabler av typen **struct Robot**, och som läser in data om två robotar till de variablerna med hjälp av funktionen **las_robot**. Därefter ska programmet visa båda robotarna med hjälp av funktionen **visa_robot**.

Uppgift 5 (11 p)

Nu ska vi skriva det riktiga programmet som beskrivs i scenariot. **main**-funktionen är redan färdig:

```
int main(void) {
    Robot robotar[MAX_ROBOTAR];
    int antal_robotar = 0;

    printf("Hur många robotar finns det? ");
    scanf("%d", &antal_robotar);
    for (int i = 0; i < antal_robotar; ++i)
        robotar[i] = las_robot();

    int arena[MAX_X][MAX_Y];
    for (int x = 0; x < MAX_X; ++x)
        for (int y = 0; y < MAX_Y; ++y)
            arena[x][y] = 0;

    kolla_namnen(robotar, antal_robotar);
    placera_ut(robotar, antal_robotar, arena);
    kolla_arenan(arena);

    return 0;
}
```

a) (4p) Skriv funktionen **kolla_namnen**. Den ska ta två argument: en array med robotposter, och ett heltal som anger hur många robotar som finns i den arrayen. Funktionen ska kontrollera om två robotar har samma namn, och den ska skriva ut nummer och namn för alla par av robotar med samma namn.

b) (3p) Skriv funktionen **placera_ut**, som går igenom robotarna i arrayen. Den tar tre argument: en array med robotar, ett heltal som anger hur många robotar som finns i den arrayen, och en matris (dvs tvådimensionell array) med heltal som beskriver arenan som robotarna finns på. Elementen i matrisen ska ange hur många robotar som finns på den positionen. Funktionen ska gå igenom robotarna i arrayen, och för varje robot ska den öka antalet robotar på motsvarande plats på arenan med ett.

c) (3p) Skriv funktionen **kolla_arenan**, som går igenom arena-matrisen, och för varje position där det finns mer än en robot ska den skriva ut koordinaterna och antalet robotar på den positionen.

d) (1p) Vi antar att både **MAX_X** och **MAX_Y** är **100**. Med vårt program ovan, går det verkligen att ha koordinater upp till **100**, och till exempel ställa en robot på positionen **x = 17, y = 100**? Om inte, förklara varför!

Uppgift 6 (10 p)

Det blir jobbigt att skriva in alla robotarna för hand varje gång vi startar programmet, så vi vill kunna spara robotarna på en fil.

a) (4p) Skriv funktionen **spara_robotar**, som sparar data om alla robotarna i en robot-array på en fil. Funktionen ska själv öppna filen med robotarna, och stänga den. Om filen inte kan öppnas, ska ett felmeddelande skrivas ut och programmet avslutas. Man ska kunna skicka en array med robotposter till funktionen, så den kan anropas från **main**-funktionen i uppgiften ovan.

b) (1p) Visa hur anropet till **spara_robotar** från **main** ser ut, om vi vill spara robotarna i arrayen **robotar**.

c) (4p) Vi måste också kunna läsa in robotarna från filen, så skriv funktionen **hemta_robotar**, som läser in data om robotar från en fil. Funktionen ska själv öppna filen med robotarna, och stänga den. Om filen inte kan öppnas, ska ett felmeddelande skrivas ut och programmet avslutas. Funktionen ska kunna anropas från **main**-funktionen i uppgiften ovan, så den ska alltså lägga de inlästa robotarna i en array med robotposter.

d) (1p) Visa hur anropet till **hemta_robotar** från **main** ser ut, om vi vill läsa in robotar från filen till arrayen **robotar**.
