



LÖSNINGSFÖRSLAG

TENTAMEN

PROGRAMMERINGSTEKNIK TEKNISK FYSIK A

4P

Datum	:	000315
Tid	:	9 - 15
Hjälpmedel	:	Inga
Antal uppgifter	:	9
Totalpoäng	:	40 (halva poängtalet krävs normalt för godkänt)

- Börja varje uppgift på nytt papper.
- Skriv ditt namn och uppgiftens nummer på varje papper.
- Skriv bara på ena sidan av papperet.
- Sortera dina papper efter uppgiftsnummer.
- Kryssa för de uppgifter du lämnar in.
- Lösningarna skall vara snyggt och prydligt nedskrivna. Tankegången skall vara lätt att följa. Alla antaganden som inte är uppenbara skall redovisas.
- C-kod krävs där implementation uttryckligen anges.
- Sist bland papperen finns en kursvärderingsblankett. Ta med den hem och fyll i den, byt den sedan mot en rättad skrivning vid tentamensgenomgången.

Tips!

- Försök på alla uppgifter! Observera att uppgifterna inte nödvändigtvis är ordnade efter svårighetsgrad.
- Uppgifter kan vara felformulerade, fråga om du är osäker eller tycker att något verkar konstigt.
- Redovisa dina resonemang så är det lättare att bortse från skrivfel.
- Om du inte kommer ihåg den exakta syntaxen, så gör ett antagande och redovisa detta.

- *Det är viktigt att du löser den givna uppgiften!*

Lycka till!

Uppgift 1 (2 p)

- a) Av vilken anledning kan man få problem om man skiftar en signed int åt vänster ?
- b) Bitmönster i minnet är väldigt maskinberoende. Kan man anta något om hur många bitar som finns i en signed int, och i så fall hur många ?

- a) *Teckenbiten längst till vänster kan byta värde - talet byter tecken*
- b) *I ANSI-C är en int (och en unsigned int) minst 16 bitar*

Uppgift 2 (2 p)

Förklara skillnaden mellan strängkopieringsfunktionerna strcpy och strncpy, och motivera varför det normalt är bättre att använda den senare av de två.

strcpy kopierar hela strängen, oavsett om den ryms i den mottagande strängen. strncpy har en parameter som anger max antal tecken som kan kopieras, på så sätt kan man försäkra sig om att inte för många tecken kopieras.

Uppgift 3 (5 p)

- a) Vad representerar variablerna stdin, stdout resp stderr ?
- b) Vilken typ har de ?
- c) scanf (och fscanf) kan få problem om de förväntar sig ett tal men det som finns att läsa är bokstäver. Nämn ett sätt att komma runt detta problem om din uppgift är att läsa in ett heltal från tangentbordet.
- d) gets(char * str) läser in en sträng från tangentbordet men bör inte användas på grund av att ett allvarligt problem kan uppstå. Vad kan hända ?
- e) printf tar som argument dels en formatsträng, dels noll eller flera argument som skall skrivas ut. Vad händer om det finns fler sådana argument än som behövs enligt formatsträngen ?

- a) *De representerar tre automatiskt öppnade filer, kopplade till resp tangentbord, skärm och skärm (för felutskriften oftast).*
- b) *FILE **
- c) *Läs in en sträng och konvertera den till ett tal*
- d) *Det finns ingen maxgräns för hur många tecken som kan läsas in -> det går alltid att mata in fler tecken än vad som är reserverade i str*
- e) *Alla argument evalueras, men endast de som behövs används (från vänster), resten ignoreras.*

Uppgift 4 (3 p)

- a) Varför är det bra att det finns standarder för programmeringsspråk ?
- b) Ofta delas stora programsystem upp i ett antal headerfiler (.h) och motsvarande källkodsfiler (.c). Varför ?
- c) I c måste alla använda funktioner vara deklarerade med en prototyp innan de används i ett anrop. Varför ?

- a) Program kan flyttas mellan olika maskiner och olika kompilatorer och ändå fungera på samma sätt. Dessutom underlättar det för programmerare när de skall läsa andras program.
- b) Inkapsling: Användaren av en rutin behöver inte veta hur den är implementerad, bara 'interfacet', dvs .h-filen. Programmet kan skrivas av flera personer samtidigt. Endast de filer som ändrats behöver kompileras om innan länkningen (sparar mycket tid vid stora system).
- c) För att kompilatorn skall kunna kontrollera typ, antal och ordning på parametrarna vid anropet.

Uppgift 5 (5 + 3 p)

a) Implementera (= skriv c-kod för) en rekursiv funktion `int summa(int start, int slut)` som summerar alla heltal mellan startvärdet och slutvärdet, inklusive start och slut. Observera att funktionen måste klara av de fall då `start = slut`, och då gäller att `summan = start`. Dessutom måste funktionen fungera på ett rimligt sätt om `start > slut`. Du får själv bestämma vad som är rimligt, men du måste motivera ditt val.

b) Implementera motsvarande funktion iterativt.

```
#include <stdio.h>
#include <stdlib.h>
```

```
int rsumma(int x, int y)
{
    if (x > y) return 0;           /* ett alternativ, även byt plats på x, y är ok */
    if (x == y) return x;
    return x + y + rsumma(x + 1, y - 1); /* flera varianter finns */
}
```

```
int isumma(int x, int y)
{
    int summa = 0, i;

    if (x > y) return 0;
    for (i = x; i <= y; i++)
        summa = summa + i;
    return summa;
}
```

```
/* testprogram, behövs ej i lösningen */
int main(void)
{
```

```
printf("%d %d", rsumma(1, 5), isumma(1, 5));  
}
```

Uppgift 6 (5 p)

Förklara vad följande begrepp är och vad de används till:

- a) preprocessor
- b) .h - fil
- c) #include
- d) länkning
- e) typedef

- a) *Ett program som behandlar källkoden innan kompilatorn ser den. Läser kommandon som börjar med '#'.*
- b) *En headerfil som normalt innehåller de prototyper och typdefinitioner som behövs för att rutinerna i motsvarande .c-fil skall kunna användas utan tillgång till källkoden.*
- c) *Ett preprocessorkommando som betyder att filen vars namn följer på kommandot kopieras in i källkoden.*
- d) *Efter kompilering fogas olika programdelar och bibliotek samman till en körbar fil av länkaren.*
- e) *Tillåter införandet av egna namn på egendefinierade typer, exempelvis på strukturer.*

Uppgift 7 (6 p)

En digital bild är uppbyggd av bildpunkter, sk pixels. Varje pixel innehåller i detta fall ett tal i intervallet 0 till 63 som beskriver ljusintensiteten i punkten (0 = svart, 63 = helt vitt). En bild lagras som heltal i en matris, som i detta fall alltid har storleken 100 x 100.

Implementera en funktion

```
klassificera(int bild[][100], int* granser, int* klasser, int  
            antGranser)
```

som klassificerar pixlarna i matrisen på så sätt för varje pixel bestäms det vilket intervall den tillhör, och dess värde byts ut mot ett värde som gäller för hela intervallet.

Gränsvärdena och motsvarande klasser finns som parametrar till funktionen, lagrade i två fält. Det ena fältet heter granser och specificerar övre gräns för varje intervall, det andra fältet heter klasser och specificerar det nya pixelvärdet för alla pixlar i motsvarande intervall. Parametern antGranser anger hur långa fälten granser och klasser är.

Ett exempel på fält med längden 5 (båda fälten är alltid lika långa):

```
int granser[] = { 20, 30, 40, 50, 55 };
int klasser[] = { 5, 10, 40, 45, 60 };
```

Detta innebär att alla pixlar med värden 0 - 20 byts ut mot värdet 5, alla pixlar med värden 21 - 30 byts ut mot 10 osv, 51 - 55 blir 60, och resten (56 - 63) sätts till maxvärdet = 63.

Funktionen skall kunna användas i följande programsegment:

```
int bild[100][100];
int granser[] = { 20, 30, 40, 50, 55 };
int klasser[] = { 5, 10, 40, 45, 60 };
...
(initiering av bild - matrisen)
klassificera(bild, granser, klasser, 5);
```

Din funktion skall naturligtvis vara så generell att den fungerar för vilka längder som helst på fälten granser och klasser, inte bara längden 5.

```
#include <stdio.h>

void klassificera(int bild[][10], int* granser, int* klasser, int
antGranser)
{
    int i, j, index;
    int komihag;

    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
            {
                index = 0;
                while ((bild[i][j] > granser[index]) && (index < antGranser))
                    {
                        index++;
                    }
                if (index == antGranser)
                    bild[i][j] = 63;
                else
                    bild[i][j] = klasser[index];
            }
}

/* testprogram, behövs ej i lösningen */
int main(void)
{
    int bild[10][10];
    int granser[] = { 20, 30, 40, 50, 55 };
    int klasser[] = { 5, 10, 40, 45, 60 };
    int i, j;

    for (i = 0; i < 10; i++)
        for (j = 0; j < 10; j++)
```

```
    bild[i][j] = i * j;

for (i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
        printf("%d ", bild[i][j]);
    printf("\n");
}
printf("\n");

klassificera(bild, granser, klasser, 5);

for (i = 0; i < 10; i++)
{
    for (j = 0; j < 10; j++)
        printf("%d ", bild[i][j]);
    printf("\n");
}
}
```

Uppgift 8 (5 p)

En stack är en ADT som ofta används som temporär mellanlagringsplats. Skriv en algoritm (dvs beskriv stegvis med ord) som vänder på texten i en sträng. Stacken har de vanliga funktionerna för att lägga till ett element, ta bort ett element, kontrollera om stacken är tom och att initiera den. Indata till din algoritm är en sträng *Str*, och när algoritmen är klar skall resultatet ligga i samma sträng. Observera att du **inte** skall skriva någon c-kod här !

1. *Initiera stacken*
2. *Initiera ett index till värdet 0*
3. *Så länge index < längden av Str utför steg 3.1, 3.2*
 - 3.1. *Placera tecknet från position 'index' i Str på stacken*
 - 3.2. *Öka index med 1*
4. *Sätt index till 0*
5. *Så länge stacken inte är tom utför 5.1, 5.2*
 - 5.1. *Ta översta tecknet från stacken och placera det i position 'index' i Str*
 - 5.2. *Öka index med 1*
6. *Klart*

Uppgift 9 (4 p)

Beskriv vad utskriften blir om följande program körs, och förklara hur parametrarna överföres till och från varje funktion:

```
#include <stdio.h>
#include <stdlib.h>

void funk1(int a, int b)
{ a = 2 + b; }

void funk2(int *a, int *b)
```

```
{ a = 2 + b; }

void funk3(int *a, int *b)
{ *a = 2 + (*b); }

int funk4(int *a, int b)
{ return *a + b; }

int main(void)
{
    int a = 3;
    int b = 4;
    int c;

    funk1(a, b);
    printf("Efter anrop av funk1: a = %d, b = %d \n", a, b);

    a = 3;
    b = 4;
    funk2(&a, &b);
    printf("Efter anrop av funk2: a = %d, b = %d \n", a, b);

    a = 3;
    b = 4;
    funk3(&a, &b);
    printf("Efter anrop av funk3: a = %d, b = %d \n", a, b);

    a = 3;
    b = 4;
    c = funk4(&a, b);
    printf("Efter anrop av funk4: a = %d, b = %d, c = %d \n", a, b, c);

    return 0;
}
```

3 4 - call by value, funktionen modifierar bara lokala kopior av argumenten

3 4 - call by value av pekarna, funktionen modifierar bara lokala kopior av pekarna

6 4 - call by value av pekarna, funktionen modifierar det pekarna pekar på, dvs det blir egentligen call by reference

3 4 7 - funktionen returnerar summan av vad första parametern pekar på plus den andra parametern

Slut på uppgifter