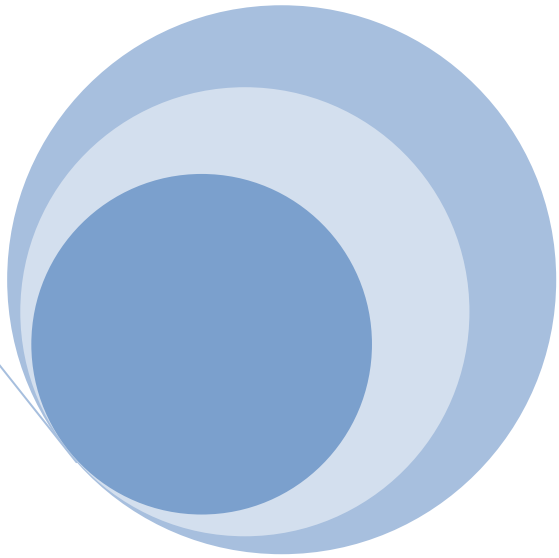
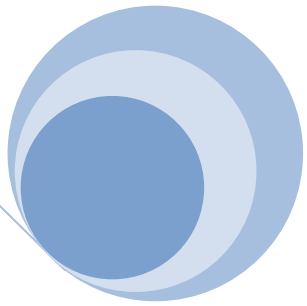


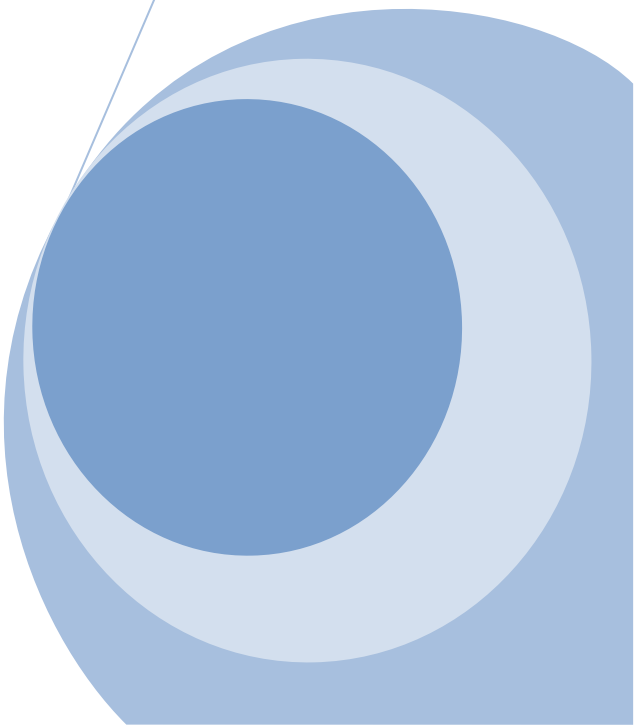


Lärobok
Programmering 1 (C++)

Nybörjarprogrammering

En enkel start på din inlärning av den sköna programmeringskonsten. I boken arbetar vi med C++, men boken innehåller mycket generell kunskap.

morgan.augustsson
2010-09-02



Inledning

Som programmerare får man ofta från andra ickeprogrammerare höra, jag begriper inte hur du har kunnat lära dig all kod utantill. Alla dessa konstiga krumelurer och lustiga tecken!

För de allra flesta så avskräcker första kontakten med programmeringskod och för de flesta så kommer dataprogrammering att förbli ett mysterium förbehållet ett fåtal underliga figurer att behärska.

Jag talar ganska ofta med mina kollegor på den skola, där jag arbetar, och framför allt så slås jag av likheten med de bekymmer som mattelärarna möter i sin undervisning. Många elever har av intet illa menande föräldrar eller andra närstående blivit itutade att matte är svårt, näst intill omöjligt att lära sig.

Jag ska börja den här läroboken i programmering med att slå hål på ett antal myter och dumma vanföreställningar.

1. Vi programmerare kan inte en massa kod utantill. Vi har tillägnat oss en metod och vi har hjälpligt lärt oss att behärska ett eller flera programmeringsspråk med alla deras grammatiska regler. När man talar om programmeringsspråk talar man visserligen inte om språkliga regler som grammatiska regler utan om språkets syntax, ett annat uttryck, men det betyder precis samma sak. I övrigt så är vi tacksamma för att det finns manualer och information på nätet.
2. Det är faktiskt inte så jättesvårt att lära sig grunderna i programmering. Det är helt enkelt ett särskilt sätt att tänka på. Man får nog säga att det här sättet att tänka på är betydligt mer begränsat, egentligen strikt, än vad tankeverksamhet vanligtvis är.
3. Har man lärt sig att programmera i ett språk, så är det som regel inte så mycket arbete med att lära sig ett nytt språk, Programmeringsprocessen är densamma och de allra flesta moment återfinns på ett eller annat sätt i de flesta programmeringsspråk. Jämför till exempel med den språkintresserade, som har goda kunskaper i Latin. Det är inte så svårt för vederbörande att lära de språk, som ingår i den latinska språkgruppen.

Allra helst så skulle en nybörjarbok i programmering vara helt språkoberoende. Då kunde man koncentrera sig på grunderna i programmeringskonsten och glömma bort de specifika egenskaper, som trots allt finns i de olika språken.

Nackdelen med ett sådant angreppssätt är att boken blir alltför teoretisk för målgruppen, ungdomar i gymnasieskolan. Inte så att ungdomar per automatik är mindre teoretiskt begåvade än äldre människor, men ungdomar har större krav på meningsfullhet. Man vill se att det händer någonting på datorskärmen, när man har skrivit ett program.

Jag har därför tänkt använda mig av C++ i de exempel som återges i den här boken. *Det kan tänkas att jag för ordningens skulle också skriver en variant av boken, där exemplen ges i Java.* Orsaken till mitt språkval är att syntaxen är någorlunda intuitiv och inte så svår att förstå. Det är förhållandevis lätt att komma igång att programmera i C++.

Programmeringsprocessen i C++

När man programmerar, så skriver man instruktioner till datorn. För att den här boken ska bli begriplig för oss svenskar, så är det en bra idé att boken är skriven på svenska. Om jag hade valt mandarin som språk för boken, så hade det fungerat bra i Kina, men betydligt sämre i Sverige, Man kan alltså slå fast, att om människor ska kunna kommunicera med varandra, så är ett gemensamt språk en förutsättning för detta. Förhållandet är precis detsamma med datorerna. För att kunna skriva instruktioner till en dator, så måste man göra det på ett språk som datorn förstår. Datorns hjärna, processorn, förstår inte vare sig Svenska, Kinesiska eller C++ heller för den delen. Processorn förstår sig enbart på ettor och nollor s.k. binärt språk. Ibland kallas det språk som förstås av processorn för maskinkod. Det hela blir inte lättare av att olika typer av processorer förstår olika typer av maskinkod.

Att skriva maskinkod är jättesvårt. Vad betyder egentligen 0010 0001 osv.? I datorernas barndom förekom säkert att dessa programmerades med maskinkod. I vissa fall var det så att det fanns brytare på datorerna och 1 betydde brytare påslagen och 0 brytare avslagen.

Numer så programmerar man i ett språk, som mer påminner om det mänskliga språket, t.ex. C++. Sedan använder man en programvara, kompilator, som översätter den kod man skrivit till maskinkod. När man programmerar i C++ tillkommer ytterligare ett moment nämligen länkningen. Efter en lyckad kompilering erhåller man en eller flera objektsfiler dessa länkas i nästa steg samman till en exekverbar fil. Det sista momentet återfinns inte i alla programmeringsspråk. I t.ex. Java länkas ett programs resurser samman under exekveringen gång.

Det betyder för dig som ska följa exemplen i den här boken att du bör ha en dator med en installerad C++ kompilator.

Att installera och komma igång

Det finns mängder med förnämliga C++ kompilatorer på marknaden. De flesta av dessa finns integrerade i ett s.k. IDE, Integrated Development Environment. Det innebär att man har kompilator, texteditor med syntaxhjälp samt länkare på ett och samma ställe. Jag vill rekommendera dig att använda en sådan programvara.

Jag kommer att använda DevCPP, som kan laddas ner gratis från nätet på http://www.bloodshed.com . Sedan har vi förstås Microsofts Visual Studio. Då måste man förstås köpa en licens. Det finns en lite enklare gratisversion av Visual Studio, som heter Visual Studio Express . När den här boken skrevs, så gällde följande adress: http://www.microsoft.com/express/Downloads/ .

Jag tror, att om du väljer att arbeta i något annat än DevCPP, så ska det nog gå bra, även om exemplen i den här boken är skrivna för DevCPP.

Vad är ett program?

Ja, ska man lära sig programmera, så är det naturligtvis av största vikt att man förstår vad ett program är för något. Du har säkert använt mängder av dataprogram, nyttoprogram, spel osv. *Det brukar för övrigt tidigt i läroböckerna framhållas att program är lika med operativsystem och allting annat är applikationer. Det brukar snart glömmas bort och så fortsätter man att tala om program. Den här boken är inget undantag, jag kommer också att tala om program.*

Man skulle kunna definiera ett program som, **instruktioner och de data som instruktionerna arbetar med.**

Jag tycker att du ska lägga definitionen ovan på minnet, eftersom den kommer att löpa som en röd tråd genom hela boken.

Instruktioner

Du blir väl besviken, om jag hävdar att processorn är ganska enfaldig och korkad. Den klarar inte av särskilt avancerade saker. Den kan lägga ihop två tal, multiplicera, dividera, subtrahera och göra jämförelser av två tal. Det som verkligen imponerar är mängden operationer och med vilken fart processorn löser sina uppgifter.

Att de operationer, som processorn kan utföra, är enkla medför, att man som programmerare måste bryta ner de problem, som man har tänkt sig ska lösas av datorn, till så små operationer som processorn klarar av. Denna nedbrytning av problem kommer också att vara ett bärande inslag i denna bok.

Data

Strängt taget så finns det väl två typer av data, **indata** och **utdata**. Indata är de data, som instruktionerna i programmet ska arbeta med och göra om till utdata.

De data, som instruktionerna arbetar med kan komma från många olika källor. Det kan finnas hårdkodat i programmets källkod, det kan vara data användaren har matat in via tangentbordet, det kan vara data som hämtats från en databas, osv.

Frågan som dryftas ibland är, vad är det för skillnad på data och information? Enligt författarens och många andras mening är information tolkad data.

När data används i ett program, så bearbetas de i form av variabler. Därför så kommer nästa avsnitt helt följdriktigt att handla om variabler och datatyper.

Variabler och datatyper

En variabel är en platshållare för data av någon typ och som uttrycket variabel antyder, så kan värdet på data variera. På nästa rad kan du se en variabel.

```
int data;
```

Variabeln har namnet data och är av datatypen int. Egentligen så är namnet data ett alias för en primärminnesadress. Ramminnet i datorn är uppdelat i minnesceller av viss storlek, för det mesta fyra bytes. Varje sådan cell har en adress. *Vi återkommer till primärminnesadresser, när vi kikar*

närmare på en variabeltyp, som kallas för pekare . Vilken adress i minnet variabeln data har, spelar inte någon roll, utan när variabeln deklareras så tilldelas den en primärminnesadress av operativsystemet.

Vidare så är variabeln data av typen int. Int är ett heltal, dvs. tal utan decimaler. Hur stor plats den upptar i minnet beror lite på datorns beskaffenhet, men 4 bytes är ganska vanligt.

Man kan också deklarera flera variabler av samma typ på en rad.

```
int tal,nummer,antalglassar;
```

På nästa rad, så kan du se ett exempel på hur en tilldelning går till.

data=5;

Ett alternativ till detta kan vara att tilldela variabeln ett värde redan från början.

```
int data=5;
```

Visst finns det fler datatyper och jag ska något översiktligt gå igenom dessa nu. Jag kommer inte att gå så djupt in på, hur data representeras i datorn. Men något måste ändå sägas om, hur stor plats de olika datatyperna ockuperar i datorns minne. Detta är nämligen datorberoende.

Behöver man veta, hur stor plats en viss datatyp t.ex. int upptar, så kan man skriva cout<<sizeof(int). På min dator skrevs 4 ut. Det innebär att på min dator så tar en int upp 4 bytes eller 32 bitar. En bit är den minsta enheten. Den kan ha värdet 0 eller 1. En byte är 8 bitar.

Heltalstyper

Alla heltalstyper är förstå till för att hantera heltal, dvs. tal utan decimaler. Alla heltalstyper i C++ finns i två varianter, signed och unsigned. Om datatypen är signed, så kan den hålla både positiva och negativa värden. Man kan skriva signed int data, men det behövs inte, eftersom heltalstyperna är signed default. Unsigned innebär att datatypen endast kan hålla positiva värden. Orsaken till att man kan behöva använda unsigned är helt enkelt, att man kan lagra större värden i en unsigned datatyp. Man deklarerar och tilldelar en unsigned int på följande sätt: unsigned int data=65000. Vi kommer inte att använda unsigned datatyper i den här boken, men det är likväl bra att känna till att de finns.

Här en liten sammanställning

Heltalstyper	
Typer som kan ha både positiva och negativa värden. Man behöver inte skriva signed framför dessa, men man kan göra det.	Typer, som endast kan ha positiva värden. Man måste skriva unsigned i deklarationen.
signed char (tecken, jag behandlar dessa under egen rubrik)	**unsigned char**
short int (Det räcker med att man skriver short)	**unsigned shor t int** (Det räcker med att man skriver unsigned short)
int	**unsigned int**
long int (Det räcker med att man skriver long)	**unsigned long int** (Det räcker med att man skriver unsigned long)

bool

Det finns ytterligare en datatyp, som väl närmast får anses vara en heltalstyp, nämligen bool (boolean). Det är i C/C++ sammanhang en ganska ny datatyp. Programmerare i språket C använde inte bool, utan man använde talen 0 respektive 1,för att få värdena sant eller falskt. Numer används istället datatypen bool, som kan ha två värden, sant eller falskt.

char

Med char representerar man tecken, både vanliga skrivtecken och styrtecken av olika slag. Att då char skulle vara en heltalstyp kan kanske förvåna. Det blir inte så svårt att förstå, om man vet att tecknen återfinns i en teckentabell. Det finns två vanliga typer, ASCII 7 och ASCII 8. Talet efter ASCII anger om teckentabellen är sjubitars eller åttabitars. Det högsta tal man kan lagra på sju bitar är 128. Det innebär att det finns plats för 128 tecken. Tyvärr för oss nordbor och andra icke engelsktalande människor, så finns det inte plats för å, ä och ö i ASCCI7. I ASCCI 8 finns det plats för 256 olika tecken och där finns det plats för å, ä och ö.

I ASCCItabellerna har varje plats ett index. På plats nummer 65 har vi bokstaven A och på plats 66 har vi B osv. Det är detta som är orsaken till att char inte bara är tecken utan också heltal.

När man deklarerar och initierar en variabel av typen char i ett program, så kan man skriva på följande sätt: char c='a'. Observera att runt tecknet har vi enkla citattecken. För att skriva ut variabeln kan vi skriva, cout<<c. Vi får då ut ett a på skärmen. Om vi ökar variabeln c med 1, vilket ju borde fungera, eftersom char är en heltalstyp, så borde vi erhålla nästa tecken i ASCIItabellen. c++; cout<<c; borde då rimligen ge b på skärmen. Vi gör ett litet testprogram, för att pröva vår hypotes.

chartest.cpp

```
#include<iostream>
#include<iodos.h>
using namespace std;
int main()
{
char c='a';
    cout<<c<<endl;
    c++;
    cout<<c<<endl;

    system("PAUSE");
        return EXIT_SUCCESS;
}
```

Programmet gav mycket riktigt det resultat vi hade tänkt oss.

```
a

b

Tryck på en valfri tangent för att fortsätta...
```

Flyttal – decimaltal

Det är inte så sällan, som man för att lösa vissa problem måste använda sig av decimaltal. Dessa brukar i programmeringssammanhang kallas för flyttal. Det beror på att decimaldelen liksom flyter.

Vad som sades ovan, när det gällde storleken på datatyperna, gäller också här, dvs. lagringsutrymmet är datorberoende. Det kan i sammanhanget berättas, att det är en ganska komplicerad process att lagra decimaltal i datorns minne. Översatt till det binära talsystemet, så kallas den första delen av decimaltalet för mantissa och decimaldelen för exponent.

Vad som är viktigt för dig, som programmerare att känna till, är att de olika datatyperna har olika exakthet. Vanligtvis brukar man hellre använda double än float, eftersom precisionen i double är bättre än i float.

Här visas en sammanställning över flyttalsstyperna.

- float
- double
- long double

Konstanter

En specialvariant på variabler är variabler, som man inte kan ändra värdet på, kosntanter. Det får till följd att man alltid tilldelar konstanten ett värde redan vid deklarationen. En konstant kan naturligtvis vara av alla de datatyper, som vanliga variabler. En bra vana är att stava konstanterna med gemener (stora bokstäver). Då ser man med en gång att det är en konstant man har att göra med. En deklaration av några konstant er kan se ut som på följande rader.

```
const int MAX=12;

const double PI= 3.14159265358979323846264338327795

/*Observera att punkt och inte komma används som
decimalavskiljare.*/
```

Typkonverteringar

Ibland, så behöver en datatyp omvandlas till en annan datatyp. Det kan ske på två sätt, dels automatiskt dels explicit (betyder att man programmeringsmässigt faktiskt gör en omvandling). En ganska vanlig omvandling är från ett decimaltal till ett heltal. Ett uttryck, som du kommer att möta i programmeringslitteraturen är casting, vilket betyder typkonvertering (typomvandling).

Automatisk typomvandlig

Automatiskt typomvandlig sker, där kompilatorn kan göra så. Till exempel så är en omvandling från double till int något, som kan utföras av kompilatorn. Vi ska kika på några exempel.

```
int tal = 5.7; //Går bra. Decimaldelen kapas och tal får värdet 5

double a = 2.1;

double b = 5.2;

int tal= a * b;//Går bra. Tal får värdet 10. Decimaldelarna kapas
```

För att sammanfatta de ovanstående kodraderna, så kan man säga att kompilatorn omvandlar värdena i högerledet till den typ, som finns i vänsterledet.

En automatisk typomvandling liknande den, som vi kan se ovan är den, som sker, om man anropar en funktion med en double, när funktionen ska ha en int som argument. *Förstå du inte orden funktion och argument, så kommer dessa att behandlas i kapitlet om funktioner.*

Funktionen Max ska ha två heltal som argument.

```
double a = 2.1;

double b =5.2;

int greatest = Max(a,b); //Går bra. greatest kommer att få värdet 5.

/*Ett tal kan omvandlas till en bool (true/false).*/

int a = 0;

int b = 1;

        if(a)

            cout<<"a = sant";

        else

            cout<<"a = falskt";

        if(b)

            cout<<"b = sant";

       else

            cout<<"b = falskt";


/*Kommer att ge utskriften, a = falskt b = sant. Vid en
typomvandling från tal till bool, så blir talet 0 falskt.*/
```

Explicit typomvandling

Man kan programmeringsmässigt tvinga fram en typomvandling. Det finns två olika syntaxer för detta.

Den första formen har C++ fått med sig från C. Det är för övrigt den formen, som för det mesta används i denna bok. *Hrm. författaren är gammalmodig!*

```
int a;
double b=34.2;
  a = (int) b; //Man skriver den typ man vill ha inom en parentes
char c;
   c=(char)a;// Man skriver den typ man vill ha inom en parentes
```

Den andra, lite nyare formen, ser ut som ett funktionsanrop.

```
int a;
double b=34.2;
        a = int(b); //Här har vi helt enkelt flyttat på parentesen
char c;
        c= char(a); //Här har vi helt enkelt flyttat på parentesen
```

Avrundning typomvandling

Avsnitten ovan om typomvandling är inte uttömmande. En del återstår, men jag väljer att behandla dessa i respektive sammanhang.

Automatiska typomvandlingar kan i vissa fall ställa till med problem. Och problemet består just i det förhållandet, att typomvandlingen sker automatiskt. Man måste tänka på, att vid en konvertering från en double/float till ett heltal, så tappar man decimaldelen. Det kan få förödande konsekvenser för viktiga beräkningar. För att råda bot mot detta, rekommenderas att du är väldigt noga vid val av datatyper i dina program. Skulle decimaldelarna vara viktiga för programmets funktionalitet, så bör du välja att använda double eller float.

Operatorer

När jag beskrev vad ett program är i ett föregående avsnitt, så skrev jag att processorn klarar av de fyra räknesätten och att den kunde jämföra olika värden. I den färdiga problemlösarbeskrivningen, algoritmen, när man har kommit så långt ner i uppdelningen av ett problem att processorn klarar av att lösa det, så har man behov av operatorer, som motsvarar processorns operationer. Det finns två stora huvudgrupper av operationer, aritmetiska och logiska.

> *Man kan också göra en uppdelning i unära och binära operatorer. Unära operatorer är operatorer med en operand och binära operatorer är operatorer med två operander. Operanden är det tecken i ett uttryck, som inte är en operator. Det kan till exempel vara en variabel. Jag tänker inte fördjupa mig i den sistnämnda uppdelningen, utan jag kommer möjligen att peka på den, när det blir aktuell.*

Aritmetiska operatorer

= tilldelningsoperator. Observera att man inte kan använda ett likamed för jämförelser, utan användningsområdet är tilldelning.

```
int a = 23;
```

a++ Uppräkningsoperator, som ökar operandens värde med ett. Man åstadkomma samma sak med a = a +1. Värdet av a++ är a:s gamla värde.

++a Samma som ovan, med den skillnaden att värdet av uttrycker är a:s nya värde.

```
a++;  //Ökar a:s värde med ett

++a; //Ökar a:s värde med ett

a = a +1; //Ökar a:s värde med ett
```

a-- Nedräkningsoperator, som minskar a:s värde med ett. Värdet av uttrycket är a:s gamla värde. Man kan uppnå samma syfte med a = a-1;

--a // Samma som ovan med den skillnaden att värdet av uttrycket är a:s nya värde.

```
a--; //Minskar a:s värde med ett

--a; //Minskar a:s värde med ett

a = 1-1; //Minskar a:s värde med ett.
```

* multiplikation

+ addition

- subtraction

/ division

% modulus

```
a  *= 34; //Kortform för a = a * 34

a += 5; //Kortform för a = a + 5

a -= a; //Kortform för a = a – a

a/=5; /* Kortform för a = a/5. Dividerar man heltal, så får man
         heltalsdivision annars, så blir det
         en vanlig division.*/

a%=5 /* Ger resten vid heltalsdivision. 10 delat på 3 ger 3,3333.
        Om man rundar av 3,333 till ett heltal, så får man
        3. 3 gånger 3 = 9. 10 – 9 = 1. Det innebär att 10 % 3 ger
1.*/
```

Att tänka på är att multiplikation och division utförs före addition och subtraktion i sammansatta uttryck. Om två operatorer har samma prioritet, så ges prioritet från vänster till höger. 5.0+3.0/2.0*2.0 ger 8, eftersom 3.0 delat på 2 är lika med 1.5. 1.5 gånger 2 är lika med 3 och 3 plus 5

är lika med 8. Mitt tips till dig som nybörjare i programmering är att styra prioritetsordningen med hjälp av parenteser. Om man skriver (2+4)/(4-1), så får man 2, eftersom uttrycken inom parenteserna utvärderas först. Som du kanske känner igen från matematiken jobbar man sig från den/de inre parentesen/parenteserna och utåt.

Logiska och jämförelseoperatorer

Logiska operatorer är den typ av operatorer, som används vid jämförelser i olika villkorsuttryck. Man kan sätta samman ganska långa och komplicerade uttryck. Mitt tips till dig är att försöks förenkla uttrycken så mycket som möjligt.

```
if(a == b) //Om a har samma värde som b, så returneras sant.

if(a >= b) //Om a är store eller lika med b, så returneras sant.

if(a <= b) //Om a är mindre eller lika med b, så returneras sant.

! //Skiljt från

if(a != 5) //Om a inte är lika med 5, så returneras sant.

||//eller

if(a==b || a==c)/* Om a är lika med b eller a är lika med c, så
                   returneras sant */

&& //Och

if(a>=3 && b != 2)/* Om a är större eller lika med 3 och b inte
                     inte är lika med 2.
```

Vårt första program

Eftersom vi nu vet vad variabler och konstanter är, så kanske vi skulle kunna snickra ihop vårt första program.

Skriv in koden nedan i din editor och kompilera sedan. Skulle det vara något syntaxfel i ditt program, så kommer kompilatorn att skriva ut ett felmeddelande. Försök hitta felet och kompilera igen. Tyvärr, så kan kompilatorernas felmeddelande vara en aning kryptiska, så det kan ta lite tid innan man lär sig att se vad det är som är fel. Jag presenterar först koden, innan jag kommenterar den ytterligare.

one.cpp

```
#include <cstdlib>

#include <iostream>

using namespace std;

int main()

{

int data;
```

```
    data=5;
    cout<<data<<endl;
    data=7;
    cout<<data<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

På de två första raderna görs två inkluderingar av bibliotek, som finns I standarduppsättningen av C++. I det här fallet inkluderas de för att vi ska kunna skriva ut någonting på skärmen och för att se till att kommandotolken, som programmet körs i inte ska stängas ner direkt, när programmet avslutas.

Sedan kommer uttrycket using namespace std. Du behöver inte bry dig om detta ett på ett tag. Jag återkommer till det längre fram i boken.

Sedan följer följande konstruktion:

```
int main()
{
    return EXIT_SUCCESS;
}
```

Detta är en funktion, som heter main. Efter funktionsnamnet finns en tom parentes . På nästa rad återfinns en krullparentes, som öppnar upp funktionskroppen, som avslutas med en krullparentes åt andra hållet. I det här fallet heter funktionen main. När programmet exekveras, så letar operativsystemet efter en funktion, som heter main. Så man skulle kunna säga att i funktionen main ligger programmet. Uttrycket return EXIT_SUCCESS betyder att programmet avslutas på ett lyckat sätt och att operativsystemet underrätas om detta. EXIT_SUCCESS är en konstant . *En konstant är som en variabel, som man inte kan ändra på.*

Raden cout<<data<<endl innebär att värdet på data skrivs ut skärmen. << är ett speciellt funktionsanrop, som vi återkommer till.

Koden i programmet exekveras uppifrån och ner, rad för rad, ungefär som man läser en bok. Denna exekveringsordning kallas för sekvens.

Du kanske också funderar över varför raderna ser tabulerade ut. Raderna är inskjutna med tabbtangenten. Ett annat ord för tabulering är indentering. Det är för övrigt det ord, som används i programmeringslitteraturen. Det är inte något måste att indentera, men det underlättar läsandet av och felsökande i koden och det är en vana, som starkt rekommenderas. Ta dig tid, att indentera din kod, du och andra kommer att tjäna på det!

Varje rad i C++ avslutas med ett semikolon, om det inte handlar om ett funktionshuvud, villkor eller någon iteration (loop) av något slag.

I programmet tilldelas en variabel, som heter data, och är av typen heltal värdet 5. Sedan skrivs värdet på data ut. Sedan sker en sker en ny tilldelning till data. Den får nu värdet 7. Detta skrivs också ut på skärmen. Till sist så har programmet körts färdigt och avslutas. Vi får följande utskrift på skärmen.

```
5

7

Tryck på en valfri tangent för att fortsätta...
```

Man kan precis som i matematiken utföra operationer på variablerna. Titta på exemplet nedan.

Vårt andra program operationer på variabler

two.cpp

```
#include <cstdlib>
#include <iostream>

#include<math.h>

using namespace std;

int main()
{
int data;
    data=5;
    cout<<data<<endl;
    //data är lika med data + data, dvs. 10
    data+=data;
    cout<<data<<endl;
    //värdet på data ökas med ett. Detta är kort för data = data +1
    data++;
    cout<<data<<endl;
    //värdet på data minskas med ett.
```

```
    data--;
    cout<<data<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Det har tillkommit lite kod i programmet. Till exempel så förekommer kommentarer i koden. När man skriver // , så kommer det som skrivs efter detta inte att kompileras. Detta gäller för en rad. Vill man skriva kommentarer som sträcker sig över flera rader, så skriver man på följande sätt:

```
/*
          Detta är en kommentar,
          som sträcker sig över
          flera rader.
*/
```

Värdet på data ändras I programmet flera gånger och skrivs ut. Jag har med avsikt använt kortformer för de matematiska operationerna. Här nedan följer en kortare sammanställning av dessa.

```
int a = 45; //deklaration och tilldelning
a´= 34; //tilldelning
a++; //Värdet på a ökas med 1
a--;//Värdet på a minskas med 1
a += 24 ;//Samma sak som a = a+24
a -= a; //Samma sak som a = a-a
a/=23;//Samma sak som a = a / 23;
a *= a; //Samma sak som a = a*a;
```

Angående uppräkning och nedräkning med 1, C++, C--, så kan nämnas att själva språket C++ är en uppräkning med 1. Det är en uppgradering av programmeringsspråket C.

Nå, hur gick det nu med vårt program. Om du lyckades kompilera programmet, så får du en utskrift vid programkörningen enligt nedan:

```
5
10
11
```

```
10
Tryck på en valfri tangent för att fortsätta...
```

Vad skulle hända om, man försökte tilldela en variabel av typen int (heltal) ett decimalvärde. Finns ett bra sätt att testa detta på? Vi gör ett program såklart!

two_a.cpp

```
#include <cstdlib>
#include <iostream>


#include<math.h>


using namespace std;


int main()
{
int data;
    data=5.4;
    cout<<data<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Man får faktiskt inget kompileringsfel, men en varning. Tydligen så kommer vårt program att försöka hantera felet på något sätt. Vi tittar på utskriften.

```
5
Tryck på en valfri tangent för att fortsätta...
```

Tydligt är att programmet löste uppgiften, genom att helt enkelt ta bort decimaldelen. Hade det nu varit så, att programmets utdata hade varit väldigt viktigt, så hade förlusten av decimaldelen kunnat ge till resultat ett felaktigt program.

Algoritmer Inledning

Rubriken algoritmer kommer också att återkomma flera gånger i boken. Här kommer jag att presentera en översikt samt några enklare exempelprogram.

Tidigare konstaterades att ett program består instruktioner och de data, som instruktionerna arbetar med. Vi har också konstaterat att de problem vi vill lösa måste delas upp i så enkla delar, att de kan lösas av en processor.

När man sedan ska skriva en problemlösarbeskrivning till datorn, så gör man det i form av en algoritm. Så i programmeringssammanhang så kan man definiera en algoritm, som en problemlösarbeskrivning.

Det finns fyra olika typiska delar i algoritmen. Dessa delar kan också ses som olika sätt att exekvera kod.

1.	Sekvens – kod exekveras rad för rad
2.	Selektion – programmet gör vägval i koden
3.	Iteration – upprepningar, nedräkningar och uppräkningar
4.	Rekursion – funktioner, som direkt eller indirekt anropar sig själva. (Behandlas längre fram i boken)

Sekvens

Sekvens innebär att kod exekveras uppifrån och ner, rad för rad. Ungefär som man brukar läsa en bok eller tidning.

Selektion

Selektion innebär att man gör vägval i exekveringen. Exempelvis, om du har fyllt arton, så kan du ta körkort annars inte. Eller om mängden äpplen är större än 10, så skriver programmet ut ”Äppelpaj” på skärmen, annars så skrivs, ”plocka mer!”, ut.

I alla moderna programmeringsspråk finns språkkonstruktioner, som används till att göra vägval. Dessa brukar kallas för if, else if och else. Om vi översätter exemplet ovan till kod, så skulle det kunna se ut som nedan.

```
int apples = användarens inmatning;
     if(apples > 10)
        cout<<"Baka äpplepaj"<<endl;
    else
        cout<<"Plocka mer äpplen"<<endl;
```

Först så tilldelas heltalsvariabeln apples värdet, användarens inmatning. Hur det går till ska jag strax visa i ett litet exempelprogram. Orsaken till att jag valt ett engelskt uttryck, apples, som variabelnamn, beror på, att man inte kan använda våra svenska bokstäver, å, ä och ö i C++.

If(apples>10) betyder, om antalet äpplen är större än tio, så exekveras raden, som står under villkoret, dvs. cout<<”Baka äpplepaj!”<<endl. Kom ihåg att det bara är en rad under villkoret, som exekveras. Har man behov av att exekvera mer än en rad efter ett villkor, så måste man omsluta dessa rader med krullparanteser. Se nedan.

```
          if(apples>10)
```

```
        {
                cout<<”Baka äpplepaj”<<endl;
                cout<<”Vispa till lite vaniljsås!”<<endl;
        }
        else
                cout<<”Plocka mer äpplen”;
```

Annars, om antalet äpplen är lika med 10 eller färre, så skrivs raden, ”Plocka mer äpplen”, ut. Även efter nyckelordet else exekveras bara en rad. Det innebär att, om man har behov att exekvera fler rader under else, så omsluter man dessa rader med krullparanteser. Ska vi ta och göra ett riktigt program av detta nu då?

Exempel selection med if och else

apples.cpp

```
#include <cstdlib>
#include <iostream>

#include<math.h>

using namespace std;

int main()
{
/* Om man inte tilldelar en variabel ett
   värde vid deklaration, så blir variabelns
   värde obestämbart. Därför kan det vara en god
   vana att tilldela variablerna ett värde, som i
   det här fallet 0. */
int apples = 0;
/*På nästa rad ges en fråga till användaren. */
    cout<<"Hur många äpplen har du plockat?"<<endl;
/*På nästa rad ges användaren en möjlighet att mata in
  ett svar på frågan. Det går till på det viset att
  användaren skriver ett tal och trycker [Enter].
```

```
  Då kommer variabeln apples att tilldelas ett värde
  Från användarens inmatning.*/
    cin>>apples;
    if(apples > 10)
        cout<<"Baka äpplepaj"<<endl;
    else
        cout<<"Plocka mer äpplen"<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
Hur mÕnga applen har du plockat?
11
Baka õpplepaj
Tryck på en valfri tangent för att fortsätta...
Hur mÕnga applen har du plockat?
2
Plocka mer õpplen
Tryck på en valfri tangent för att fortsätta...
```

Här ovan visas utskrifter från två körningar. Av utskrifterna på skärmen att döma, så har vårt program fungerat som vi hade tänkt oss. Det finns dock två saker att ta upp i det här sammanhanget.

Förmodligen har du fått lika konstiga utskrifter av å. ä och ö, som jag. Det beror egentligen på att kommandotolken i Windows inte kan hantera dessa bokstäver. Det finns lösningar på det problemet och jag kommer att presentera en sådan längre fram i boken. Till dess, så skulle man kunna tänka sig, antingen att man skriver texterna på engelska eller att man byter ut å mot a, ö mot o och ä mot a. Jag kommer fortsättningsvis att välja det senare alternativet.

Det finns ingen som helst kontroll av att användaren skriver ett giltigt tal. Denne kanske svarar, ”trettiotvå”. Längre fram i boken, så kommer jag att visa, hur en sådan kontroll skulle kunna gå till.

Exempel med if, else if och else

Vi kastar oss rätt in i exemplet.

temperatur.cpp

```
#include<iostream>


using namespace std;


int main()
{
int temperatur;
cout<<"Vilken temperatur ar det ute?"<<endl;
cin>>temperatur;


    if(temperatur>25)
           cout<<"Det ar varmt ute. Ta av dig jackan!"<<endl;
    else if(temperatur>18)
           cout<<"Du behover inte ta pa dig nagot extra!"<<endl;
    else
          cout<<"Kallt ute. Ta pa dig nagot extra!"<<endl;
              system("PAUSE");
       return EXIT_SUCCESS;
}
```

Programmet frågar användaren efter temperaturen ute. Användarens inmatning lagras i variabeln temperatur. Sedan fattas ett beslut, om vilket meddelande, som ska skrivas ut till användaren. Laborera gärna med att förändra värdena.

En körning av programmet kan se ut som nedan:

```
Vilken temperatur ar det ute?
19
Du behover inte ta pa dig nagot extra!
Tryck på en valfri tangent för att fortsätta...
```

När man använder sig av if, if else och else är det lätt hänt att det blir fel. Skulle programmet inte fungera som du tänkt dig, så kontrollera att villkoren sitter rätt. Om man t.ex. hade börjat ovanstående program med raden if(temperatur > 0), så hade det villkoret svalt alla temperaturer

över 0. Då hade nästa villkor else if(temperatur>8) varit fullständigt meningslöst. Det är lätt hänt med andra ord, att man kommer fel i logiken.

Switch ett alternativ till if och else

Skulle det vara så att man behöver fatta många beslut med if och else, så kan det bli ganska rörigt och svårt att komma rätt. Ett alternativ kan då vara att använda konstruktionen med switch(uttryck som ska utvärderas). Jag tänker i vanlig ordning visa ett exempel.

switch.cpp

```
#include<iostream>
using namespace std;

int main()
{
          int tal=0;
          cout<<"ge mig ett tal!"<<endl;

          switch(tal)
          {
          case 1:
          cout<<"Du är en stjarna"<<endl;
          break;
          case 2:
          cout<<"Du borde borja studera!"<<endl;
          break;
          case 3:
          cout<<"Vad ska vi gora med dig da?"<<endl;
          break;
          default:
          cout<<"Siffran ar hogre an tre"<<endl;
          }
           system("PAUSE");
          return EXIT_SUCCESS;}
```

Som du kan se i programmet ovan, så initieras en heltalsvariabel, som heter tal. Sedan uppmanas användaren att mata in värdet på tal. Sedan så ska olika saker skrivas ut på skärmen beroende på värdet på tal. Man skulle naturligtvis kunna skriva if(tal==1) cout<<”Du är en stjärna”<<endl; osv. Men här används en switchsats. Man kan beskriva det som att i switch görs en utvärdering av variabeln tal. Skulle värdet vara 1, dvs. case: 1, så skrivs ”Du är en stjärna”, ut på skärmen. Observera att case värde avslutas med ett kolon och inte ett semikolon. Sist i varje casefall, så finns uttrycker break, vilket får utvärderingen av uttrycket (tal) att avstanna. Underlåter man att skriva break sist, så finns risken att värdet (tal) rinner igenom hela switchkonstruktionen. Sista case-etiketten är default, vilket innebär att alla värden, som inte fastnar i något av villkoren ovanför hamnar här.

En körning av programmet kan se ut så här:

```
ge mig ett tal!

3

Vad ska vi gora med dig da?

Tryck på en valfri tangent för att fortsätta...
```

Exempel på selektion med villkor? sant : falskt

Sist i det här avsnittet om selektion ska vi titta på en något ålderdomlig konstruktion, som inte desto mindre är väldigt användbar. Det är en något krånglig syntax, men min förhoppning är att jag ska kunna göra den begriplig. Även om du själv väljer att inte använda den, så kommer du förr eller stöta på den, när du läser kod i böcker eller i program, som någon annan har skrivit.

Det är ett uttryck som skriv på en rad och innehåller följande: **villlkor ? (om villkoret är sant, så händer det här): (om villkoret är falskt, så händer det här)**.

Kanske blir det lite lättare att förstå, om jag visar ett litet program.

```
#include<iostream>
#include<iodos.h>

using namespace std;

int main()
{
          int godisdagar=0;
          const int MAX_DAGAR=1;

          cout<<"hur manga dagar i veckan ater du godis?"<<endl;
```

```
          cin>>godisdagar;


godisdagar > max_dagar ? cout<<"Du ater for mycket godis!" <<endl:
cout<<"Din konsumtion ar nog okey!"<<endl;


          system("PAUSE");


          return EXIT_SUCCESS;
}
```

Godis är som bekant gott, men för mycket av den varan kan ge bekymmer med tänderna, så man skulle kanske begränsa godisätandet till en dag i veckan.

I programmet frågar vi helt enkelt användaren, hur många dagar i veckan denne äter godis, och lagrar uppgiften i variabeln godisdagar. Vi har en konstant, som vi kallar för MAX_DAGAR. Den håller gränsvärdet för det antal dagar i veckan man bör äta godis.

Sedan kommer det nya uttrycket, där vi först prövar om godisdagar är större än antalet MAX_DAGAR. Skulle så vara fallet, så exekveras det första uttrycket efter frågetecknet. Skulle så inte vara fallet, så exekveras uttrycket direkt efter kolonet. Tyvärr, så har jag av utrymmesskäl varit tvungen att lägga ett radbryt mitt i uttrycket, som helst bör skrivas på en rad.

En liten provkörning gav följande resultat:

```
hur manga dagar i veckan ater du godis?
7
Du ater for mycket godis!
Tryck på en valfri tangent för att fortsätta...
```

Iteration (while, do-while och for)

Att itererera innebär att man upprepar någonting. Vi ska kika på ett par exempel på iterationer. Vi ska börja med en iteration, som ser ut enligt nedan:

```
          while(någonting är sant)
          {
            gör någonting.
          }
```

Whilesatsen utför någonting så länge som ett villkor är sant. Även i det här fallet är det så att det är första raden under villkoret som exekveras. Behöver man exekvera fler rader under villkoret, så måste man omgärda dessa med krullparanteser. Självklart, så ska vi kika på ett exempelprogram.

while.cpp

```
#include<iostream>
#include<iodos.h>

using namespace std;

int main()
{
          int smorgas = 0;
          const int MATT=10;
          while(smorgas<matt)
          {
                    smorgas++;
                    cout<<"Ater smorgas nr. "<<smorgas<<endl;
          }
          cout<<"Nu ar jag matt"<<endl;

    system("PAUSE");
                        return EXIT_SUCCESS;
}
```

En variabel av heltalstyp benämnd smorgas initieras till värdet 1. Är man mätt, så är man det när man har ätit tio smörgåsar, därför använder vi en konstant av heltalstyp, som heter MATT. Så länge som smorgas är mindre än MATT, så äter vi upp ytterligare en smorgas (räknar upp smorgas med ett) . Eftersom värdet på smorgas kommer att bli samma som på MATT, så kommer whileloopen att avslutas, eftersom villkoret returnerar falskt.

En körning av programmet ser ut som nedan:

```
Ater smorgas nr. 1

Ater smorgas nr. 2

Ater smorgas nr. 3

Ater smorgas nr. 4

Ater smorgas nr. 5

Ater smorgas nr. 6
```

```
Ater smorgas nr. 7

Ater smorgas nr. 8

Ater smorgas nr. 9

Ater smorgas nr. 10

Nu ar jag matt

Tryck på en valfri tangent för att fortsätta...
```

Iteration med do-while

Nu ska vi kika på en iteration, som påminner väldigt mycket om while, nämligen do-while. Skillnaden mellan de båda är att satserna i do-while alltid exekveras minst en gång, eftersom villkorsprövningen ligger sist. En while-loop där villkoret returnerar falskt redan från början exekveras inte alls. Konstruktionen med do-while ser ut som nedan:

```
do
{
          Gör någonting
}
while(villkor);
```

Självklart, så ska vi göra ett litet exempelprogram, där vi testar en do-whileloop.

do_while.cpp

```
#include<iostream>
#include<iodos.h>

using namespace std;

int main()
{
          int smorgas = 1;
          const int MATT=10;

          do
          {
           smorgas++;
```

```
        cout<<"Ater smorgas nr. "<<smorgas<<endl;
    }while(smorgas < MATT);
          cout<<"Nu ar jag matt"<<endl;

    system("PAUSE");
        return EXIT_SUCCESS;
}
```

Jag tror att du förstår hur programmet fungerar utan att jag förklarar I detalj. En programkörning ger precis samma resultat, som i förra exemplet. Men vi ska skriva om programmet en aning, för att se vad som händer, om vi redan har ätit 11 smörgåsar, när vi kör igång programmet.

do_while_b.cpp

```
#include<iostream>
#include<iodos.h>

using namespace std;

int main()
{
          int smorgas = 11; //Ändras till 11
          const int MATT=10;

          do
          {
           smorgas++;
           cout<<"Ater smorgas nr. "<<smorgas<<endl;
           }while(smorgas < MATT);
          cout<<"Nu ar jag matt"<<endl;

    system("PAUSE");
        return EXIT_SUCCESS;
}
```

En programkörning ger följande utskrift:

```
Ater smorgas nr. 12

Nu ar jag matt

Tryck på en valfri tangent för att fortsätta...
```

Vad ska vi dra för slutsats av detta nu då? Att det är olämpligt att använda do-while, eftersom man riskerar att få ont i magen? Ja kanske det, men det viktigaste är att du förstår, att do-while satserna alltid exekveras en gång innan villkorsprövningen sker. Någon gång så kan det vara just precis vad du behöver för att lösa ett problem.

Iteration med for

Nu ska vi titta på en lite mer komplicerad, men desto mer användbar iteration, den så kallade for-loopen. Mallen för for-loopen ser ut som nedan:

```
for(variabeldeklaration/tilldelning; villkor; uppräkning/nedräkning)
```

Efter nyckelordet for ska det vara en parentes. I denna parentes ska det vara tre avsnitt åtskiljda av två semikolon. I första avsnittet deklarerar eller tilldelar man en eller flera variabler värden. I avsnitt två, så görs en villkorsprövning av något slag. I tredje avsnittet, så brukar man normalt sett göra uppräkningar eller nedräkningar av de variabler, som återfinns i första avsnittet. Det är inte nödvändigt att verkligen skriva någonting i alla avsnitt, men då måste de lämnas tomma. Till exempel, så ger följande uttryck for(; ;) en evighetsloop. Det blir kanske lättare, om jag ger ett exempel i ett litet program.

for_ett.cpp

```
#include<iostream>
using namespace std;


int main()
{
    for(int i=0;i< 5;i++)
        cout<<"i="<<i<<endl;
     system("PAUSE");
        return EXIT_SUCCESS;
}
```

I for-loopen deklareras en variabel av heltalstyp. Det är också där som exekveringen av uttrycket startar. I for-loopens nästa avsnitt görs en prövning av om i är mindre än 5. Om villkoret returnerar sant, så körs en rad under uttrycket. Vi får utskriften ”i=0”. Sedan hoppas exekveringen till sista

avsnittet i uttrycket och uppräkning av i sker. I första svängen, så får i värdet 1. Sedan sker en ny villkorsprövning, skulle den prövningen resultera i att sant returneras, så exekveras åter raden under uttrycket och ”i=1” skrivs ut. Så håller for-loopen på, ända tills i får värdet 5, då villkorsprövningen returnerar falskt, vilket resulterar i att for-loopen avbryts. Vi erhåller nedanstående utskrift från en körning av programmet.

```
i=0
i=1
i=2
i=3
i=4
Tryck på en valfri tangent för att fortsätta...
```

Man kan deklarera och tilldela fler variabler i samma for-loop och man är inte alls bunden till att räkna upp variablerna med ett. Man är faktiskt inte alls bunden till att räkna upp, man kan även räkna ner. Vi ska kika på ett lite mer sammansatt exempel för att demonstrerar vad jag sagt.

for_two-.cpp

```
#include<iostream>
#include<iodos.h>

using namespace std;

int main()
{
    for(int i=0,j=20;i< 10;i+=2,j-=3)
        cout<<"i="<<i<<" och j = "<<j<<endl;

    system("PAUSE");
        return EXIT_SUCCESS;
}
```

Som du kan se i första avsnittet i for-loopen, så deklareras och tilldelas mer än en variabel. Den andra variabeln j, skapas efter ett komma. Lägg märke till att datatypen har utelämnats, den blir automatiskt av typen int. I sista avsnittet så räknas i upp med 2 och j ned med tre. Observera att de olika operationerna åtskiljs av ett komma. Programmet ger följande utskrift på skärmen.

```
i=0 och j = 20
```

```
i=2 och j = 17

i=4 och j = 14

i=6 och j = 11

i=8 och j = 8

Tryck på en valfri tangent för att fortsätta...
```

Vi kommer tillbaka till for-loopen väldigt många gånger i den här boken, eftersom den är så användbar. I all synnerhet är den användbar, när man arbetar med fält, eller arrayer, som de också kallas. Arrayer är en särskild typ av variabler, som vi ska kika närmare på längre fram. Men jag tänkte i alla fall avsluta det här avsnittet om iterationer med ett exempel, där jag lägger en for-loop inuti en annan for-loop, för att skriva ut en enklare multiplikationstabell.

multiplikation.cpp

```
#include<iostream>
#include<iodos.h>
using namespace std;
int main()
{
    for(int i=1;i<11;i++)
    {
        for(int j=1;j<11;j++)
        {
            cout<<i<<" * "<<j<<" = "<<(i*j)<<endl;
        }
    }

    system("PAUSE");
        return EXIT_SUCCESS;
}
```

Det är nog inte så svårt att se, hur det fungerar. Första gången, som den yttre loopen går in i den inre, så har i värdet 1 och j värdet 1. 1*1 är som bekant 1. Sedan räknas j upp med 1 och vi får 1*2, vilket är 2. Så för att sammanfatta, första gången i går in i j får vi ettans gångertabell, andra gången får vi tvåans osv.

Programmet skriver ut följande tjusiga multiplikationstabell . *Jag har tagit bort en del av utskriften för att spara utrymme.*

```
1 * 1 = 1

1 * 2 = 2

1 * 3 = 3

1 * 4 = 4

1 * 5 = 5

1 * 6 = 6

1 * 7 = 7

1 * 8 = 8

1 * 9 = 9

1 * 10 = 10

2 * 1 = 2

2 * 2 = 4

2 * 3 = 6

2 * 4 = 8

2 * 5 = 10

2 * 6 = 12

2 * 7 = 14

2 * 8 = 16

2 * 9 = 18

2 * 10 = 20

3 * 1 = 3

3 * 2 = 6

3 * 3 = 9

Tryck på en valfri tangent för att fortsätta...
```

Rekursion

För att kunna förstå sig på rekursion, så måste man också ha grundläggande kunskap om en annan av programmeringens byggstenar, funktioner. Jag återkommer därför med rekursion längre fram i boken.

Man kan avbryta en iteration med nyckelordet break

Med hjälp av nyckelordet break, så kan man alltid avbryta en iteration . Vi kikar på ett exempel. Det handlar om en for-loop, som ska skriva ut talen 0 till och med 9. Men, om värdet på variabeln i är jämt delbart med 5, så avslutas loopen med break. Variabeln i får startvärdet 1, eftersom om jag hade tilldelat den värdet 0, så hade 0 % 5 gett 0 och loopen hade avslutats med en gång.

Det här exemplet är förstås helt meningslöst, eftersom for-loopen alltid avslutas i förtid. När man vill använda konstruktionen med break i ett riktigt program, så beror det på, att man vet, att det skulle kunna inträffa något, som gör det nödvändigt eller önskvärt att avbryta iterationen i förtid.

break_me.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
srand(time(0));
    for(int i=1;i<10;i++)
    {
        cout<<"i="<<i<<endl;
        if(i % 5 == 0)
                break;
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift:

```
i=0
i=1
i=2
i=3
i=4
```

```
Tryck på en valfri tangent för att fortsätta...
```

Att manipulera en iteration med continue

Låt oss anta att vi skapar en iteration, t.ex. en for-loop, som skriver ut tal 0 tom 25. Den skulle kunna se ut så här:

```
for(int i=0;i<30;i+=5)
{
     cout<<"i="<<i<<endl;
}
```

Programmet ger mycket riktigt utskriften:

```
i=0
i=5
i=10
i=15
i=20
i=25
Tryck på en valfri tangent för att fortsätta...
```

Om man nu till exempel tror att siffran 15 för otur med sig, eller om man är allergisk mot talet, så kan man lägga till ett villkor i for-loopen, som innebär att exekveringen av iterationen i och för sig fortsätter, men det som sker efter continue, hoppas över.

```
for(int i=0;i<30;i+=5)
{
          if(i==15)
            continue;
        cout<<"i="<<i<<endl;
}
```

Programmet ger nu utskriften:

```
i=0
i=5
i=10
i=20
i=25
```

```
Tryck på en valfri tangent för att fortsätta...
```

Det var väl kanske inte det mest praktiska exemplet på continue, som har skådats, men det är i alla fall enkelt och åskådliggörande.

Att skapa algoritmer metodik (1)– exempel

Innan vi kör igång på allvar med programmerandet, så ska vi försöka komma tillrätta med utskrifterna av våra svenska tecken å, ä och ö. Professor Jan Skansholm har skrivit ett bibliotek, som man kan ladda ner från nätet, http://www.cse.chalmers.se/~skanshol/cpp_eng/files/iodos.h

Spara ner filen som iodos.h i DevCPPinstallationens includemap. *Det ska det i det här sammanhanget nämnas att Jan Skansholm bland mycket annat också har skrivit en bok i C++, som heter C++Direkt.*

Vi gör ett litet testprogram.

```
#include<iostream>
#include<iodos.h> // Ska läggas till
using namespace std;
int main()
{
    dos_console(); //Initierar biblioteket
    cout<<"Välkommen på er alla!"<<endl;

    system("PAUSE");
        return EXIT_SUCCESS;
}
```

Fungerar det som det ska, så erhåller vi följande vackra utskrift.

```
Välkommen på er alla!

Tryck på en valfri tangent för att fortsätta...
```

När nu detta är avklarat, så ska vi ge oss i kast med det som utlovas i rubriken, nämligen att skapa algoritmer, som löser ett och ett anna t problem.

När man programmerar, så har man som regel en uppgift, som man vill att datorn ska utföra. Eftersom datorns processor inte klarar av att utföra annat än väldigt enkla operationer, så måste man bryta ner uppgiften eller problemet till enklare delproblem/deluppgifter.

VI ska gå igenom några enkla exempel.

Tärningar och Snake Eyes

När man slår två tärningar, så händer det ibland att man får två ettor och det brukar benämnas snakeeyes. Hur ofta man får det varierar förstås! Men viss vore det väl intressant att få veta hur många gånger det händer på tusen slag. Det är bara att plocka fram tärningarna, penna, papper och en skaplig portion tålamod. Men tänk om man skulle vilja veta hur många gånger det händer på tio tusen slag. Samma rekvisita men tålamodet måste vara ännu större.

Men nu är ju du programmerare och då är det väl bara att skriva ett program och låta datorn göra jobbet. Vi ska börja med att ställa två vikiga frågor.

1. Vad har vi för data att arbeta med?
2. Vilka instruktioner bör datorn få, för att kunna lösa uppgiften?

Vad har vi för data att arbeta med?

Ja, för det första, så måste vi få fram två tärningsslag. Det vore väl lämpligt att lagra dessa i två heltalsvariabler. Dessa värden måste förstås slumpas fram. Sedan måste vi ha någon variabel, som håller reda på hur många gånger de två tärningsslagen blir ettor. Det är väl den variabeln, som kan betraktas som utdata, eftersom det är den uppgiften vi presenterar för användaren. Till sist, så måste vi ha en variabel, som håller reda på, hur många gånger vi har slagit tärningarna. Åh. Ja visst ja, någon konstant kanske, som anger hur många tärningsslag, som ska slås.

Vilka instruktioner är lämpliga?

Jag, eftersom vi nu är familjära med algoritmens grundstenar, så ska vi kanske börja använda dessa. När någonting ska ske flera gånger, så använder man en iteration. Jag tror inte det spelar så stor roll vilken egentligen, men jag tycker att vi väljer en for-loop. Den ska förstå iterera 10 000 gånger. I for-loopen, så slumpar vi fram två tal mellan 1 och sex. Skulle båda dessa ha värdet 1, så räknar vi upp variabeln för antalet snakeeyes. När vi har itererat färdigt, så har vi antalet snakeeyes i en variabel, som vi kan presentera för användaren, genom att skriva ut dess värde på skärmen.

Sammanfattning algoritm

- Slå tärningarna
- Kolla om båda dessa har värdet ett
- Om båda tärningar har värdet 1, räkna upp variabeln antal_snake_eyes med 1.
- Om vi inte har slagit tillräcklig många gånger, så hoppar vi till första punkten igen.

Vi skulle få ett program, som ser ut enligt nedan.

snake_eyes.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
#include<iodos.h>

using namespace std;
```

```
int main()
{
  dos_console();
/*Det första som händer i programmet är
  att vi måste fixa till en ny slumptalsintervall.
  Det gör man med hjälp av srand(time(0)). Jag får säkert
  anledning att återkomma till hur slump genereras i
  i C++. Tänk dock på att datorn egentligen inte i egentlig
  mening kan slumpa fram någonting alls, utan endast kan
  efterlikna slump. Sedan är det förstår en i det närmaste
  filosofisk fråga om slump över huvud taget existerar och i
  sådana fall, hur den fungerar*/
          srand(time(0));
/*Här följer sedan deklarationer och tilldelningar
  av de variabler, som vi kom överens om skulle
  finnas i programmet.*/
int dice_one=0;
int dice_two=0;
int antal_snake_eyes=0;
const int ANTALSLAG=10000;
/* Här har vi då for-loopen, där själva
   arbetet sker. Två tärningsslag slumpas fram.
   Syntaxen för slumpgenereringen återkommer jag
   till längre ner efter koden. När man jämför två variabler,
   så används två likamedtecken. Ett likamedtecken har
   man bara vid tilldelning.
   && betyder och. Så uttryckt på svenska, så blir det
   om dice_one är lika med ett och dice_two är lika med ett,
   så räknar vi upp variabeln antal_snake_eyes med 1. */
          for(int i=0;i<ANTALSLAG;i++)
          {
                    dice_one=(rand()%6)+1;
                    dice_two=(rand()%6)+1;
                              if(dice_one==1 && dice_two==1)
                                        antal_snake_eyes++;
```

```
          }
/* Sist men inte minst, så presenterar vi resultatet, utdata
   från programmet. */
cout<<"Av "<<ANTALSLAG<<" slag så blev blev det snake eyes "<<
                              antal_snake_eyes<<" ggr."<<endl;
             system("PAUSE");
            return EXIT_SUCCESS;

}
```

Följande för oss tärningsspelare nedslående resultat gavs vid en provkörning.

```
Av 10000 slag så blev det snake eyes 298 ggr.
Tryck på en valfri tangent för att fortsätta...
```

Jo, jag hade ju lovat att säga något om slumpgenereringen. Som sagt ”äkta” slump är inte någonting, som datorn behärskar. Datorn kan dock efterlikna slump och det sker genom en oerhört komplicerad algoritm. Vi ska dock titta lite på, hur man kan slumpa fram värden och hur man kan bestämma inom vilka intervall de slumpade värdena ska ligga.

Man börjar som regel med srand(time(0)), det kallas för att man byter slumptalsintervall. Funktionen rand() fungerar utan srand. Då får man förvisso till synes slumpmässiga output, men man får samma slump varje gång man kör sitt program.

Jag har inte gått igenom funktioner ännu, men anropar man vissa funktioner, så får man någonting tillbaka. Det gäller till exempel funktionen rand()- Vi gör ett litet testprogram, där vi skriver ut returvärdet från rand några gånger.

test_rand.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
using namespace std;
int main()
{
int start = 0;
const int TIMES=10;
    srand(time(0));
```

```
        while(start < TIMES)
        {
            cout<<rand()<<endl;
            start++;
        }
              system("PAUSE");
            return EXIT_SUCCESS;

}
```

Programmet gav följande utskrift vid en körning på min dator.

```
13612
3464
16631
21651
9650
3437
2088
30885
25041
8278
Tryck på en valfri tangent för att fortsätta...
```

Man kan sammanfattningsvis säga att vi fick ut ett antal heltal. I fallet med tärningar, så ville vi ju ha ut tal i intervallet 1 till 6. För att få till det, så användes %. % är den så kallade modulusoperatorn. Om man dividerar två heltal med varandra och använder modulus istället för en vanlig division, så erhåller man det som blir över. 2 % 2 ger förstås 0, eftersom 2 är jämnt delbart med 2. Samma gäller 25 % 5.

Om man däremot utför heltalsdivisionen 11/3, så får man 3.6666 och heltalsdelen är då 3. 3 * 3 = 9. 11 – 9 = 2. Det betyder att 2 är det som blir kvar. Man kan alltså slå fast att 11 % 3 är lika med 2.

Om vi nu gör om vårt program en aning, så att vi utför en modulusoperation på det returvärde, som vi får från rand, så ska vi se om det händer något.

test_rand_two.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
#include<iodos.h>

using namespace std;
int main()
{
int start = 0;
const int TIMES=10;
    srand(time(0));

        while(start < TIMES)
        {
            cout<<rand()%6<<endl; //Nytt här
            start++;
        }

              system("PAUSE");
            return EXIT_SUCCESS;

}
```

Programmet ger nu följande utskrift.

```
4
2
5
2
0
3
```

```
4

5

3

2

Tryck på en valfri tangent för att fortsätta...
```

Studerar vi utskriften lite, så ser vi att värdena ligger från 0 till 5. Det enda vi behöver göra är att öka alla värden med 1, för att få tal, som kunde ha slagits med en tärning.

test_slump_tre.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
using namespace std;
int main()
{
int start = 0;
const int TIMES=10;
    srand(time(0));

        while(start < TIMES)
        {
            cout<<(rand()%6)+1<<endl;
            start++;
        }
                system("PAUSE");
            return EXIT_SUCCESS;

}
```

Programmet gav följande utskrift.

```
6

1
```

Ja, då hoppas jag att du har lärt dig något om slump i datorn och hur modulusoperatorn fungerar. Jag tror vi fortsättet på slump och tärningsspåret.

Hur många gånger måste datorn slumpa ett tal, för att slumpa fram just 8374

Nu när vi lärt oss, hur man kan låta datorn slumpa fram ett tal, så ska vi låta datorn slå lite fler tärningar.

Visst är det imponerande att se, hur snabbt datorn arbetar. Nu har jag låtit hjärnan slumpa fram ett tal. Det blev talet 8374. Om man nu låter datorn slumpa fram ett tal mellan 1 och 20 000, så är frågan, hur många gånger behöver den slå en tärning, för att hitta samma tal?

Vad har vi för data att arbeta med?

Vi behöver förstås ha en variabel av heltalstyp, som håller reda på, hur många gånger datorn slumpar. Denna variabels vvärde blir också utdata från programmet. Man kan använda en bool, som är falsk, så länge som 8374 inte har slumpats fram. Till data hör förstås det eller de tal, som slumpas fram. Dessa ska ha ett värde mellan 1 och 20 000.

Vilka instruktioner är lämpliga?

Här finns säkerligen ett flertal alternativ. Man skulle kunna använda en while-loop. Den ska exekvera, så länge, som en bool är falsk. I denna räknar man upp variabeln, som håller antalet slumpade tal. Sedan slumpar man ett nytt tal och jämför denna med 8374. Skulle det slumpade talet överensstämma med det sökta, så ställs boolvariabeln om till true, vilket får while-loopen att sluta exekvera. Slutligen, så presenteras resultatet på skärmen.

1. En bool kallad hittad får värdet false
2. Vi går in i en while-loop, som körs så länge som hittad är false
3. Vi slumpar ett tal mellan 1 och 20 000
4. Vi räknar upp antalet slumpade tal
5. Vi jämför det nya talet med 8374
6. Skulle det slumpade talet vara lika med 8374, får hittad värdet true. Loopen stannar av och programmet kan presenterar resultatet för användaren.

7. Skulle det slumpade talet inte vara lika med 8374, så går vi till punkt 3

slump_four.cpp

```
#include<iostream>
#include<iodos.h>
#include<ctime>
#include<stdlib.h>
using namespace std;

int main()
{
dos_console();
int number_of_trials=0;
bool hittad=false;
srand(time(0)); //Ny slumptalsintervall

                    while(!hittad)
                    {
                    number_of_trials++;
                              if(rand()%20000==8734)
                                        hittad=true;
                    }
    cout<<"Det krävdes "<<number_of_trials<<
           " slumptal för att hitta talet "<<8734<<endl;

          system("PAUSE");
            return EXIT_SUCCESS;
}
```

```
En testkörning på min dator gav följande resultat:

Det krävdes 5787 slumptal för att hitta talet 8734

Tryck på en valfri tangent för att fortsätta...
```

Det är förstås självklart att resultatet kan variera kraftigt från körning till körning. Skulle man lyckas få 8734 på ett försök, så bör man nog fundera på att köpa en lott eller så får man leta efter fel i koden.

Sex sexor i rad

Oj då! Skulle man lyckas med den bedriften, så har man antingen en oerhörd tur, eller så är man fingerfärdig. Men hur många gånger skulle man behöva slå tärningen innan man lyckas få sex sexor i rad. Nu är vi ju programmerare, så självklar så gör vi ett program, för att undersöka hur det ligger till.

Vad har vi för data att arbeta med?

Ja vi ska väl ha någon variabel, som håller reda på, hur många sexor, som slagits. Ytterligare en variabel håller reda på hur många tärningsslag, som slagits. Sedan måste vi förstås slumpa fram tärningsslagen.

Vilka instruktioner är lämpliga?

Ja, följande upplägg kanske fungerar. Vi gör en loop, som avslutas när antalet sexor i rad är sex.

1. Vi slår en tärning och räknar upp variabeln för antalet tärningsslag
2. Om tärningen är en sexa, så räknar vi upp variabeln för antalet slagna sexor med ett
3. Om det inte är en sexa, så måste vi nollställa variabeln för antalet slagna sexor, dvs. börja om.
4. Kontrollera om antalet sexor är lika med sex
5. Om antalet sexor inte är lika med sex, så hoppar vi till punkt 1, annars så går vi till nästa punkt.
6. Presentera antalet tärningsslag för användaren

sex_i_rad.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
#include<iodos.h>

using namespace std;
int main()
{
int antal_tarningsslag = 0;
int antal_sexor = 0;
    dos_console();
    srand(time(0));

    while(antal_sexor < 6)
```

```
    {
    antal_tarningsslag++;
        if( ((rand()%6)+1)==6)
        {
            antal_sexor++;
        }
        else
            antal_sexor=0;
    }

    cout<<"Datorn behövde "<< antal_tarningsslag<<
         " slag, för att slå sex sexor i rad."<<endl;
          system("PAUSE");
            return EXIT_SUCCESS;
}
```

Följande utskrift gavs vid en körning på min dator.

```
Datorn behövde 107019 slag, för att slå sex sexor i rad.

Tryck på en valfri tangent för att fortsätta...
```

Tipsvinsten och glassarna

Du hade lite tur med stryktipset i helgen. Du hade lyckats knåpa ihop en rad, som gav tio rätt, vilket gav 87 kr. Nu funderar du som bäst på, hur du ska spendera pengarna. Det finns en sak, som du inte kan motstå, din favoritglass ChockoDreams, som kostar 13,50. Nu är frågan, hur många glassar kan du köpa och hur mycket pengar blir över?

Det är klar, att man skulle kunna använda en miniräknare och räkna ut det. Man kan säkert också lösa det genom en uträkning i ett program, men jag tänkte att vi skulle använda lite av det vi lärt oss tidigare och lösa uppgiften med en iteration.

Vad har vi för data att arbeta med?

Jo, vi har ju summan pengar, som du vann på tipset och priset på glassen. Båda dessa ska vi naturligtvis lagra i varsin variabel. På något sätt, så måste vi lagra kostnaden för de glassar, som du faktiskt kan köpa. Det får bli ytterligare en variabel.

Som utdata, ska vi ha en variabel, som håller reda på antalet glassar och en annan variabel, som håller den summa pengar, som blir kvar, när köpet av glassarna är klar.

Vilka instruktioner är lämpliga?

Man skulle kunna tänka sig en while-loop. Som villkor i den, kan man ha, att så länge som kostnaden för inköpta glassar understiger tipsvinsten, så fortsätter man att köpa glassar. I while-loopen, så köper man en glass, räknar upp antalet glassar med 1 och summerar kostnaden. När while-loopen är klar, så får man resten, genom att minska tipsvinsten med kostnaden för glassarna. Sist, så presenteras resultatet för användaren.

Sammanfattning av algoritmen

1. Så länge som kostnaden för glass understiger tipsvinsten
2. Köp en ny glass
3. Räkna upp antalet inköpta glassar
4. Räkna upp kostnaden med priset för en glass
5. Om kostnaden understiger tipsvinsten gå till punkt 2, annars till nästa punkt
6. Räkna ut resten, genom att minska tipsvinsten med kostnaden
7. Skriv ut antalet glassar och resten av tipsvinsten på skärmen

tipsvinst.cpp (Ett första försök)

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
double tipsvinst=87;
double glasspris=13.50;
int antal_glassar=0;
double kostnad=0;
double rest = 0;
        while(kostnad < tipsvinst)
        {
            antal_glassar++;
            kostnad+=glasspris;

        }
        rest = tipsvinst-kostnad;
        cout<<"Antal glassar: "<<antal_glassar<<endl;
        cout<<"Kvar: "<<rest<<":kr"<<endl;
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Glöm aldrig bort att testa dina program! Ett test på min dator gav följande resultat.

```
Antal glassar: 7
Kvar: -7.5:kr
Tryck på en valfri tangent för att fortsätta...
```

Aj då! Någonting har gått fel. Om jag tolkar utskriften rätt ovan, så har jag blivit skyldig glasshandlaren 7,50 kr. Inte bra! Men vad är det som gått fel?

Om vi studerar villkoret i while-loopen, så sägs att så länge som kostnaden för glass understiger tipsvinsten, så ska jag köpa ytterligare en glass. Det är precis vad som hänt! När sista glassköpet skedde, så var min totala kostnad lägre än tipsvinsten.

Hur ska man lösa det då?

Det finns säkert flera tänkbara lösningar, men mitt förslag är, att vi inne i while-loopen kontrollerar kostnaden efter ett avslutat köp. Skulle kostnaden vara högre än tipsvinsten, så minskar vi antalet glassar med 1, räknar av kostnaden med priset för en glass och avslutar while-loopen i förtid med ett break. Det sistnämnda är väldigt viktigt, eftersom vi annars skulle hamna i en evighetsloop. Kan du lista ut varför?

tipsvinst.cpp (andra försöket)

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
double tipsvinst=87;
double glasspris=13.50;
int antal_glassar=0;
double kostnad=0;
double rest = 0;
```

```
        while(kostnad < tipsvinst)
        {
            antal_glassar++;
            kostnad+=glasspris;
            /*Ifsatsen nedan är ny */
            if(kostnad>tipsvinst)
            {
                antal_glassar--;
                kostnad-=glasspris;
                break;//avslutar while-loopen
            }
        }
        rest = tipsvinst-kostnad;
        cout<<"Antal glassar: "<<antal_glassar<<endl;
        cout<<"Kvar: "<<rest<<":kr"<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger nu följande utskrift:

```
Antal glassar: 6
Kvar: 6: kr
Tryck på en valfri tangent för att fortsätta...
```

Vilken tur att vi testade programmet! Du slapp både magont och fick en slant över. En snabb kontroll på min kalkylator visade att programmet nu fungerar korrekt.

Arrayer

Det finns en särskild typ av variabel, som det är oerhört viktigt att man förstår sig på, innan man försöker begripa sig på pekare och texthantering i C++. *Pekare är en särskild typ av variabel, som vi ska kika lite närmare på i kapitlet Pekare – primärminnesadresser och referenser.*

Array kallas också i svensk programmeringslitteratur för fält. Jag kommer att hålla mig till ordet array. En array är en indexerad behållare. En behållare, som man lagra flera värden i. Istället för att ha många variabler av samma typ i ett program, så kan man istället använda en array. Programmen blir förmodligen lite mer lättlästa, om man inte har variabelnamn som x23 eller y19.

Det finns två regler, när det gäller arrayer, som man måste hålla sig till.

- Arrayen måste ha ett bestämt antal platser
- Arrayens innehåll måste vara av samma typ

int tal[5]; är en array med plats för 5 st. element. Om man gör en liten bild av hur tal ser ut just nu, så skulle det kunna se ut så här.

adress	Värde
0	
1	
2	
3	
4	

Arrayens namn: tal
Arrayens storlek: 5
Arrayen typ: int

Vad man ska vara extra vaksam på, när man arbetar med arrayer, är att dessa är nollindexerade. Första adressen är nämligen 0. Detta kan ibland stöka till det i tankearbetet med en algoritm.

När man har skapat en array, så kan man peta in värden på de olika adresserna. Det kan t.ex. se ut som nedan:

```
tal[0] = 34;
tal[1]=tal[0];
tal[2]=tal[0]-23;
tal[3]=1100;
tal[4]=23;
tal[5]=45; //Detta är felaktigt, eftersom vi nu har 6 element i
behållaren. Vi sa, att det skulle vara 5.
```

Efter dessa rader, så ser arrayen ut som på bilden.

adress	Värde
0	34
1	34
2	11
3	1100
4	23

I många programmeringsspråk finns en inbyggd indexeringskontroll. Det innebär inte att man får något kompileringsfel, om man har försökt lägga in värden utanför arrayens bestämda storlek. Däremot, så får man ett körtidsfel. Så är inte fallet i C++. Av orsaker, som har med pekare att göra (*pekare går jag igenom längre fram*), så kan konsekvenserna bli allvarliga i ett C++ program, där man råkar komma utanför arrayen bestämda storlek. Ansvaret för att hålla sig inom ramarna ligger därför helt och hållet på programmeraren.

Man kan arbeta med arrayens element precis som med vanliga variabler. Det är väl kanske dags för ett litet exempelprogram. Utöver att du får se hur man deklarerar en array, så får du också se, hur smidigt det är att loopa igenom en array med hjälp av en for-loop.

array_1.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
using namespace std;
int main()
{
    const int SIZE=5;
    int tal[SIZE];
    tal[0] = 34;
    tal[1]=tal[0];
    tal[2]=tal[0]-23;
    tal[3]=1100;
    tal[4]=23;

        for(int i=0;i<SIZE;i++)
        {
            cout<<"tal["<<i<<"]="<<tal[i]<<endl;
        }
          system("PAUSE");
            return EXIT_SUCCESS;
}
```

Programmet gav följande illustrativa utskrift.

```
tal[0]=34
tal[1]=34
tal[2]=11
tal[3]=1100
tal[4]=23
```

```
Tryck på en valfri tangent för att fortsätta...
```

Man kan deklarera och fylla en array med innehåll i ett uttryck. Det finns två alternativ.

- int tal[5] = {3,78,12,92,11};
- int tal[] = {3,78,12,92,11}; //kompilatorskapad array

Det är inte så stor skillnad på de två uttrycken, men i andra fallet, så saknas storleken på arrayen inom hakparentesen. Grundregeln är att man alltid måste ange, vilken storlek en array ska ha. Vad som händer i fall två är, att kompilatorn går in och räknar antalet element inom krullparentesen och skriver sedan dit antalet i hakparentesen. Detta kallas för en kompilatorskapad array. Skulle det hända att du får problem med uttryck nummer två, så kan det bero på att din kompilator inte klarar av att skapa någon array. Gör helt enkelt som i första fallet. Självklart, så är det väl dags för ett litet exempelprogram.

array_2.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
using namespace std;

int main()
{
    int tal[] = {3,78,12,92,11};

        for(int i=0;i<5;i++)
        {
            cout<<"tal["<<i<<"]="<<tal[i]<<endl;
        }

        system("PAUSE");
            return EXIT_SUCCESS;
}
```

Det är inte så stor skillnad på det här programmet och det förra. Skillnaden är bara, att vi i det här exemplet använder en kompilatorskapad array. Det finns ett litet problem i den här koden, som inte känns så bra. Jag fick själv gå in i arrayen och räkna hur många element, som fanns, för att kunna iterera den i en for-loop. Så frågan är då, hur ska man på ett mer dynamiskt sätt räkna ut storleken

på en array? Jo sedan tidigare vet vi att funktionen sizeof returnerar hur många bytes någonting tar i datorns minne.

```
cout<<sizeof(int); //skriver ut 4 på min dator.

cout<<sizeof(tal) //skrev ut 20 på min dator. Det betyder att
arrayens data är 20 bytes.
```

Om man delar den totala storleken på en array i bytes med storleken i bytes på den datatyp, som finns i arrayen, så får man antalet element i arrayen. Nu prövar vi!

array_3.cpp

```
#include<iostream>

#include<ctime>

#include<stdlib.h>

using namespace std;


int main()

{

    int tal[] = {3,78,12,92,11};

    int storlek = sizeof(tal)/sizeof(int);

    cout<<"Antal element i arrayen = "<<storlek<<endl;


        for(int i=0;i<storlek;i++)

        {

            cout<<"tal["<<i<<"]="<<tal[i]<<endl;

        }

          system("PAUSE");

            return EXIT_SUCCESS;

}
```

Programmet gav följande utskrift vid en körning på min dator.

```
Antal element i arrayen = 5

tal[0]=3

tal[1]=78

tal[2]=12
```

```
tal[3]=92

tal[4]=11

Tryck på en valfri tangent för att fortsätta...
```

Jo som tur är, så verkar det fungera som tänkt.

Att lösa problem med arrayer (Algoritmer)

Vi ska gå igenom några vanliga programmeringsuppgifter, som inbegriper arrayer. Om vi har ett antal provresultat från ett matteprov i en skolklass. Hur ska vi då göra, om vi vill att datorn ska ta fram det bästa resultatet?

Vad har vi för data?

I det här fallet, så är det väl ganska givet, vad vi har för data att arbeta med. Vi lägger helt enkelt in elevernas testresultat i en array av typen double. En variabel för att lagra det högsta värdet är inte bara bra, utan helt nödvändigt. Det är för övrigt den variabel vars värde är utdata i det här programmet.

Vilka instruktioner är lämpliga?

Jo, på något sätt, så måste vi jämföra alla resultaten i arrayen, för att se vilket av resultaten, som är högst. Mitt förslag är att vi börjar med påståendet att arrayens första element är det högsta. Därför så tilldelar vi variabeln **best** värdet av arrayens första element. Sen loopar vi igenom resten av arrayen. Om vi använder en for-loop, så börjar vi alltså på 1. Sedan jämför vi varje värde med variabeln best, skulle det vara så att elementet är högre än **best**, så tilldelar vi **best** värdet av det aktuella elementet. När vi sedan har loopat igenom hela arrayen, så kommer **best** att hålla värdet för det högsta provresultatet.

Anta att första elementet i arrayen är högst och tilldela variabeln **best** detta värde

- Iterera arrayen med start från 1
- Jämföra elementets värde med **best**.
- Om element har ett högre värde än **best**, så får **best** detta värde

provresultat.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
#include<iodos.h>
using namespace std;


int main()
```

```
{
    dos_console();
    double provresultat[] = {3.5,27.2,18.2,28.2,11.0,15.5,
                             18.5,17.5,19.0};
    int storlek = sizeof(provresultat)/sizeof(double);
    cout<<"Antal provresultat = "<<storlek<<endl;
    double best = provresultat[0];
        for(int i=1;i<storlek;i++)
        {
           if(provresultat[i]>best)
                best = provresultat[i];
        }
    cout<<"Bästa resultatet är "<<best<<" poäng."<<endl;
          system("PAUSE");
            return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift:

```
Antal provresultat = 9

Bästa resultatet är 28.2 poäng.

Tryck på en valfri tangent för att fortsätta...
```

Ett program som räknar ut medelvärdet på provresultatet

Man kanske vill beräkna provresultatets medelvärde. Medelvärdet får man förstås genom att räkna samman alla provresultat och dela med antalet prov.

Vad har vi för data?

Först och främst, så har vi förstås alla provresultaten i en array. Sedan bör vi ha en variabel, som håller summan av provresultaten. Antalet prov får vi genom att räkna ut antal element i arrayen. Slutligen en variabel, som håller medelvärdet och som vi kan presentera för användaren via en skärmutskrift-

Vilka instruktioner är lämpliga?

Vi loopar igenom alla elementen med t.ex. en for-loop. Där ökar vi på variabeln, som håller summa för provresultat, med värdet på varje enskilt element. När iterationen är klar, så delar vi slutsumman med antalet prov och lagrar uppgiften i variabeln för medelvärdet. Sist men inte minst, så presenterar vi resultatet för användaren.

provresultat_medel.cpp

```
#include<iostream>
#include<ctime>
#include<stdlib.h>
#include<iodos.h>
using namespace std;

int main()
{
    dos_console();
    double provresultat[] = {3.5,27.2,18.2,28.2,11.0,15.5,
                             18.5,17.5,19.0};
    double summa_provresultat = 0;
    double medel=0;
    int storlek = sizeof(provresultat)/sizeof(double);
    cout<<"Antal provresultat = "<<storlek<<endl;

        for(int i=0;i<storlek;i++)
        {
          summa_provresultat+=provresultat[i];
        }
    medel = summa_provresultat/storlek;
    cout<<"Provets medelresultat är "<<medel<<" poäng."<<endl;

        system("PAUSE");
            return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift:

```
Antal provresultat = 9
Provets medelresultat är 17.6222 poäng.
Tryck på en valfri tangent för att fortsätta...
```

Innan vi går vidare med att skriva ett program, som tillåter inmatning från tangentbordet, vilket förmodligen skulle kännas mer användbart, så ska vi ägna oss lite åt ett problem, som blir uppenbart i det här programmet. Som du kan se, så blev utskriften av medelvärdet en aning långt, eller rättare sagt, det blev lite väl onödigt många decimaler. Vi ska tittal lite närmare på olika möjligheter att styra utskrifter längre fram i boken, men vi ska ändå redan nu åtgärda det problem, som uppstod i det här programmet.

Det finns någonting, som man kallar för utskriftsmanipulatorer. Du har redan sett en, nämligen endl. Vi har skrivit så här på många ställen i våra program, cout<<”Lite text”<<endl;, vilket då har gett ett radbryt. Vi kunde lika gärna ha skrivit cout<<”Lite text\n”, eller möjligen cout<<”lite text\r\n”; Ett n med en backslash framför betyder line feed och r med en backslash framför betyder return. Att man skriver en backslash framför beror på, att man vill att dessa tecken inte ska behandlas som vanliga tecken, utan de ska styra utskriften. Backslash och ett tecken är en s.k. flyktsekvesn. Vi kommer att titta närmare när flyktsekvenser, när vi går igenom strängar och tecken.

De utskriftsmanipulatorer man skulle kunna använda i vårt program ovan, är fixed och setprecision(2). Fixed anger att flyttal ska skrivas ut med fast form och att decimalavskiljare ska användas. Med setprecision(2) anger man, att man vill ha en utskrift med två decimalers noggrannhet. Vi får anledning återkomma till utskriftsmanipulatorer längre fram, men nu vill jag presentera, hur vi ska kunna få en bättre utskrift i vårt program ovan. Byt ut utskriftsraden med följande kod:

```
cout<<"Provets medelresultat är "<<fixed<<setprecision(2)<<medel<<" poäng."<<endl;
```

Inkludera en fil, som heter iomanip. Utskriften från programmet blev nu istället:

```
Antal provresultat = 9

Provets medelresultat är 17.62 poäng.

Tryck på en valfri tangent för att fortsätta...
```

Bättre, eller hur?

Användarens inmatning

Innan vi går vidare med en lite lyxigare variant av provresultatsprogrammet, där vi kommer att låta användaren mata in testresultaten i en array av typen double, så måste vi lösa problemet med att användaren matar in felaktiga data i programmet. Vi ska undersöka problematiken med hjälp av ett par mindre program.

inmatning_ett.cpp

```
#include <cstdlib>
#include <iostream>


using namespace std;
```

```
int main()
{
    double data;
        if(cin>>data)
            cout<<"data = "<< data<<endl;
        else
            cout<<"felaktigt nummerformat"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Som vi tidigare har sett, så innebär uttrycket cin>>data, att användarens inmatning hamnar i variabeln data. Om nu data är av typen double och användaren skriver femtiofem på tangentbordet, så kommer inte någon tilldelning att ske, eller i vart fall som helst uppstår ett fel. Det förnämliga med uttrycket sin>>data är att det returnerar falskt om inmatningen inte går att placera i variabeln.

Det kan man då utnyttja på det sätt som visas i programmet ovan.

Två körningar av programmet gav följande utskrifter:

```
skrivbord
felaktigt nummerformat
Tryck på en valfri tangent för att fortsätta...
55.6
data = 55.6
Tryck på en valfri tangent för att fortsätta...
```

I det här programmet, så får användare bara ett försök att komma rätt. Det är väl inte särskilt användarvänligt. Det vore väl bättre att låta användaren hålla på tills denne kommer rätt. Till det så behöver man någon form av iteration. Eftersom det nu är så fiffigt ordnat att uttrycket cin>>variabel returnerar falskt, om en tilldelning inte kan ske, så skulle man kunna lägga ett villkor i en whileloop, som inte avslutats innan indata är korrekt. Det skulle kunna se ut som nedan:

```
While(!(cin>>variabel))
{
          cout<<”Felaktig indata! Var god försök igen!”<<endl;
}
```

Om du studerar uttrycket, så upptäcker du förstås en nyhet. Vad i hela världen betyder (!(cin>>variabel). Utropstecknet! betyder ”skiljt från” och innebär att ett uttryck är sant, om det är falskt. T.ex. if(a != 34) betyder att om a inte är lika 34, så returnerar uttrycket sant. Vi gör väl ett litet testprogram igen då?

inmatning_two.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
    dos_console();
int age=0;
    cout<<"Hur gammal är du?"<<endl;
    while(!(cin>>age))
    {
        cout<<"Felaktigt format på inmatningen. Försök igen!";
    }
          cout<<"Du är "<<age<<" år gammal! "<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Om man nu provkör programmet och anger sin ålder på korrekt sätt, så får man följande utskrift:

```
Hur gammal är du?
27
Du är 27 år gammal!
Tryck på en valfri tangent för att fortsätta...
```

Observera att programmet inte såg att jag ljög kraftigt om min ålder! Men formatet på indata var tydligen helt korrekt. Men om jag istället hade skrivit tjugosju, så hade en underlig sak inträffat. Det hade börjat skriva ut meddelandet, ”Felaktigt format på inmatningen. Försök igen!”, hur många gånger som helst. Jag hade dessutom inte fått chansen att korrigera mitt misstag. Du kan väl pröva programmet igen och den här gången så skriver du tjugosju, som svar på frågan om din ålder. Du kan tvångsstänga programmet med [ctrl][c].

Problemet beror på att cin>> läser data från inströmmen från tangentbordet, och att i inströmmen ligger det fortfarande tecken, som cin>> kan läsa. Tecknet som ligger där är vårt enterslag, som vi avslutade den första inmatningen med.

Medicinen mot detta är två funktionsanrop, clear och ignore. clear upphäver det feltillstånd, som inströmmen har hamnat i. ignore tar två argument, det första är antalet tecken i inströmmen, som maximalt ska hoppas över. Det andra argumentet är det tecken överhoppningen ska stanna på, t.ex. enterslaget. Vill man hoppa övermaximalt 100 tecken eller fram till enterslaget, så skriver man cin.ignore(100,’\n’). Båda funktionerna, clear och ignore hör till cin. Det är därför man skriver cin punkt och funktionsnamnet.

Nu har vi ju inte gått igenom det här med funktioner ännu, men det kommer längre fram. Att funktioner hör till något objekt, t.ex. cin hör till en programmeringsmodell, somkallas objektorientering, vilket vi inte heller har gått igenom. Det kommer också att förklaras lite längre fram i boken.

Vi skriver om programmet lite.

inmatning_tre.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
    dos_console();
int age=0;
    cout<<"Hur gammal är du?"<<endl;
    while(!(cin>>age))
    {
        cout<<"Felaktig format på inmatningen. Försök igen!"<<endl;
```

```
        cin.clear();
        cin.ignore(100,'\n');
    }
    cout<<"Du är "<<age<<" år gammal! "<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En provkörning av programmet gav följande utskrift:

```
Hur gammal är du?
kaffe
Felaktig format på inmatningen. Försök igen!
34
Du är 34 år gammal!
Tryck på en valfri tangent för att fortsätta...
```

Programmet tycks ju fungera precis som tänkt.

Lärarens provprogram.

Det finns ett mindre problem, som vi måste lösa i det här programmet. Ett problem att ta hänsyn till är att vi inte på förhand kan veta, hur många elever det är som har skrivit provet. I det här programmet, så kommer vi att lösa problemet på det viset, att vi deklarerar en rejält tilltagen array. Förhoppningsvis, så behöver vi inte använd alla platserna i arrayen.

Egentligen, så hade vi behövt använda någon slags dynamiskt array. Längre fram i boken, så kommer vi att använda en sådan, men nu kör vi med en väl tilltagen array.

teachers_solution.cpp

```
#include<iostream>
#include<stdlib.h>
#include<iodos.h>
#include<iomanip>
using namespace std;
```

```
int main()
{
    dos_console();
    const int MAXRESULTAT=100;
    int faktiska_resultat=0;
    double provresultat[MAXRESULTAT];
    double summa_provresultat = 0;
    bool klar=false;
    /* Vi börjar med att hantera lärarens inmatning */

        while(!klar)
        {
        double temp=0;
        cout<<"Provresultat nr: "<<(faktiska_resultat+1);
        cout<<" -1 avslutar"<<endl;
            while(!(cin>>temp))
            {
                cout<<"Felaktigt format!"<<endl;
                cin.clear();
                cin.ignore(100,'\n');
            }
            if(temp == -1)
                klar = true;
            else
            {
                provresultat[faktiska_resultat] = temp;
                faktiska_resultat++;
            }
        }
```

```
    /* Sedan tar vi reda på vilket provresultat, som är bäst.*/
    double best = provresultat[0];
        for(int i=1;i<faktiska_resultat;i++)
        {
            if(provresultat[i]>best)
                    best=provresultat[i];
        }
    /* Nästa steg är att räkna ut provets medelresultat.*/

    double medel=0;

    cout<<"Antal provresultat = "<<faktiska_resultat<<endl;
        for(int i=0;i<faktiska_resultat;i++)
        {
          summa_provresultat+=provresultat[i];
        }
    medel = summa_provresultat/faktiska_resultat;
/*Sist men inte minst, så presenterar vi resultatet för läraren.*/
    cout<<"Provets bästa resultat är "<<best<<"
poäng."<<endl;cout<<"Provets medelresultat är
"<<fixed<<setprecision(2)<<medel<<" poäng."<<endl;

        system("PAUSE");
            return EXIT_SUCCESS;
}
```

Jag tror att du förstår allting I programmet ovan, eftersom alla delar har behandlats separat. Enda tillägget är väl egentligen att vi via variabeln faktiska_resultat håller reda på, hur många platser i arrayen vi har använt.

En provkörning av programmet gav följande utskrift.

```
Provresultat nr: 1 -1 avslutar
```

```
78

Provresultat nr: 2 -1 avslutar

23

Provresultat nr: 3 -1 avslutar

11

Provresultat nr: 4 -1 avslutar

79

Provresultat nr: 5 -1 avslutar

45

Provresultat nr: 6 -1 avslutar

39

Provresultat nr: 7 -1 avslutar

38

Provresultat nr: 8 -1 avslutar

67

Provresultat nr: 9 -1 avslutar

-1

Antal provresultat = 8

Provets bästa resultat är 79 poäng.

Provets medelresultat är 47.50 poäng.

Tryck på en valfri tangent för att fortsätta...
```

Tecken texter och strängar

I C++ finns datatypen char för att hantera tecken, både skrivtecken och styrtecken. Det finns två modeller för att behandla strängar, dels finns en standardklass, som heter string, dels den gamla csträngen. Vi ska behandla båda modellerna i den här boken och börjar med standardklassen string. Men innan dess, så ska vi undersöka med strängarnas minsta beståndsdelar, tecken.

Enstaka tecken inläsning

Det finns olika sätt att läsa in enstaka tecken från tangentbordet. Man kan använda en funktion, som heter get(tecken). När vi inledningsvis kikade på datatypen char, så skrev vi cin>>tecken, för att läsa in enstaka tecken. cin är ett slags objekt, som finns färdigt i programmet, om man inkluderar

iostream. Detsamma gäller för övrigt cout. Det är objektet cin, som har tillgång till funktionen get(tecken), så för att använda den, så skriver man cin.get(tecken). Det är med hjälp av punkten (punktoperatorn), som man kommer åt de funktioner, som cin har tillgång till.

enstaka_tecken.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
char tecken;
    cout<<"Ge mig ett tecken tack!\n";
    cin.get(tecken);
    cout<<"Tack tecknet "<<tecken<<" satt bra!\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Det finns kanske en sak i koden ovan som förvånar dig lite. Vad är \n? Jo, inte alla tecken är till för att skrivas ut på skärmen. Det finns också så kallade styrtecken. Om man vill använda ett styrtecken, så ska man skriva en backslash framför tecknet. Då begriper cout, att tecknet inte ska skrivas ut, utan någonting annat ska hända. Tecknet \n representerar ett radbryt, så i programmet ovan, så ger \n samma resultat som cout<<endl; Vi ska kika mer på flyktsekvenser om en liten stund bara.

Programmet ovan gav följande utskrift:

```
Ge mig ett tecken tack!
d
Tack tecknet d satt bra!
Tryck på en valfri tangent för att fortsätta...
```

När man matar in någonting i ett program från tangentbordet, så avslutar man inmatningen med entertangenten. Entertangenten genererar naturligtvis också ett tecken. Funktionen get(tecken) hämtar dock bara ett tecken. Men vad händer då med tecknet för entertangenten?

flera_tecken1.cpp

```
#include <cstdlib>
```

```
#include <iostream>
using namespace std;
int main()
{
char tecken;
        while(tecken != 'x')
        {
            cout<<"Ett tecken tack!\n";
            cin.get(tecken);
            cout<<"Tecknet: "<<tecken<<" satt bra!\n";
        }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Tanken är att användaren ska mata in ett tecken valfritt antal gånger. Inmatningen avslutas med tecknet x. En utskrift av programmet ger följande resultat:

```
Ett tecken tack!
M
Tecknet: M satt bra!
Ett tecken tack!
Tecknet:
 satt bra!
Ett tecken tack!
L
Tecknet: L satt bra!
Ett tecken tack!
Tecknet:
 satt bra!
Ett tecken tack!
x
```

```
Tecknet: x satt bra!

Tryck på en valfri tangent för att fortsätta...
```

Vad är nu det här då? Programmet började med att uppmana användaren att skriva ett tecken. Denne skrev ett M. Programmet svarade med,”Tecknet: M satt bra!”. Men sen händer något konstigt. Användare uppmanas återigen, men får aldrig chansen att skriva något.

Vad som händer är att cin hittar tecknet för enter och nöjer sig med det. För att få programmet att fungera som tänkt, så måste vi alltså få bort enterslaget från indataströmmen. cin har tillgång till en funktion, som heter get(), utan argument. Den funktionen returnerar sant, om det gick att läsa in ett tecken, annars falskt. (Om du vill veta mer om funktioner, så hoppar du i förväg till avsnittet om funktioner). Den första varianten get(tecken) kan man kedja ihop med andra funktioner, som hör till cin. Man kan alltså skriva, cin.get(tecken).get();. Det får till effekt att enterslaget läses in och förbrukas. Det skulle lösa vårt problem.

flera_tecken2.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
char tecken;
        while(tecken != 'x')
        {
            cout<<"Ett tecken tack!\n";
            cin.get(tecken).get(); //OBS Nytt!
            cout<<"Tecknet: "<<tecken<<" satt bra!\n";
        }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En körning av programmet på min dator gav följande resultat:

```
Ett tecken tack!
M
Tecknet: M satt bra!
Ett tecken tack!
```

```
o

Tecknet: o satt bra!

Ett tecken tack!

x

Tecknet: x satt bra!

Tryck på en valfri tangent för att fortsätta...
```

Flyktsekvenser

Som vi har sett nu i några exempel, så kan man använda styrtecken, om man skriver en backslash framför. Sekvensen \n ger ett radbryt. Vi ska kika på några fler flyktsekvenser i nästa program.

escape_sekvenser.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    cout<<"\a"; //Ger ett beep från datorn
    cout<<"TextA\tTextB"<<endl; // \t Ger en tab.
    cout<<"Rad1\nRad2"<<endl; // \t Ger ett radbryt.
    /*\r flyttar utskriftspositionen
      till början av raden.*/
    cout<<"\tMorgan\rAugustsson\n";
    /* \b backspace - utskriftspositionen
       flyttas ett steg till vänster. */
    cout<<"\tOve\bEmil\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift (bortsett från datorbeepet förstås):

```
TextA   TextB

Rad1

Rad2
```

```
Augustssonrgan
      OvEmil
Tryck på en valfri tangent för att fortsätta...
```

Tabell över några flyktsekvenser

\n	**Ny rad**
\a	**Datorns beepljud**
\b	**Backspace. Flyttar utskriftspositionen ett steg åt vänster**
\r	**Return. Flyttar utskriftspositionen längst till vänster.**
\t	**Tab**

Standardklassen string

Vi har ju inte gått igenom objektorienterad programmering, som inbegriper klasser och objekt. Det faller utanför ramarna för den här boken, men vi kommer att kika på några av de mest användbara standardklasserna. För att ha någon nytta av det här avsnittet, så skulle du kunna betrakta en klass, i det här fallet klassen string, som ett avancerat bibliotek. En standardklass är ett bibliotek, som ingår i en normal installation av C++. Vi ska kika på ett inledande exempel.

string_ett.cpp

```
#include<iostream>
#include<stdlib.h>
#include<iodos.h>
#include<string>
using namespace std;

int main()
{
    dos_console();
    string str;
    str = "Morgan Augustsson";
    cout<<str<<endl;
    str="C++ är häftigt!";
    cout<<str<<endl;
```

```
        system("PAUSE");
            return EXIT_SUCCESS;
}
```

Det första att tänka på, när man ska använda standardklassen string, är att man måste inkludera headerfilen string.

Som du kan se så deklareras en sträng på nästan samma sätt, som en vanlig variabel, string str;

Tilldelningen är inte heller särskilt komplicerad, str=”Morgan Augustsson”. Tänk dock på att den text, som ska tilldelas strängen som värde ska inneslutas av dubbla citattecken.

Man kan naturligtvis även skriva ut en sträng, cout<<str<<endl.

Programmet gav följande utskrift:

```
Morgan Augustsson
C++ är häftigt!
Tryck på en valfri tangent för att fortsätta...
```

Man kan sätta samman textsträngar med andra textsträngar med hjälp plustecknet. Det kallas med ett fint ord för att konkatenera. Med ett mer folkligt språk, så heter det, att man klistrar ihop textsträngar.

concat.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
string a ="skrattar";
string b ="bäst";
string c ="som";
```

```
string d ="sist.";
string e =a+" "+b+" "+c+" "+a+" "+d;
    cout<<e<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Det är strängvariabeln e, som skapas genom att de andra strängvariablerna konkateneras. Programmet ger följande utskrift:

```
skrattar bäst som skrattar sist.
Tryck på en valfri tangent för att fortsätta...
```

En sträng är en rad av tecken i följd. När vi lite senare behandlar de gamla csträngarna, så kommer vi att upptäcka, att en sträng i själva verket är en array med char, som har ett avslutande nolltecken. Men även i en string kan man komma enskilda tecken med hjälp av arrayindexering. Titta på nedanstående exempel.

mors_lill_olle.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
string str="Mors lilla Olle i skogen gick";

    cout<<str<<endl;
    for(int i=0;i<str.size();i++)
    {
        if(str[i]=='o' ||
```

```
            str[i]=='O' ||
                str[i]=='e')
        {
            str[i]='i';
        }
    }
    cout<<str<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Koden ovan fordrar lite förklaringar. Vi börjar med att skapa en sträng, som innehåller texten ”Mors lilla Olle i skogen gick”. Denna skrivs ut.

Sedan itererar vi igenom strängen med hjälp av en for-loop, tecken för tecken. Observera att strängen vet, hur många tecken den innehåller. Man kan helt enkelt fråga den via metoden size(). Sedan plockar vi ut enskilda tecken med hjälp av arrayindexering []. Sedan kollar vi om tecknet är lika med O, o eller e. De två piporna betyder ”eller”. Observera att, när vi gör jämförelserna, så jämför vi med char. Det betyder att man måste innesluta tecken med enkla citat. Skulle prövningen returnera sant, vilket händer om något av villkoren uppfylls, så ställer vi strängens tecken (string[i]) till ett i. Vi byter helt enkelt ut alla vokaler, utom i, till i.

Programmet gav följande utskrift:

```
Mors lilla Olle i skogen gick

Mirs lilla illi i skigin gick

Tryck på en valfri tangent för att fortsätta...
```

Visst kan man låta användare mata i strängvärden via kommandotolken. Titta på nedanstående exempel.

instring.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>


using namespace std;
```

```
int main()
{
dos_console();
string fnamn;

    cout<<"Vad heter du? ";
    cin>>fnamn;
    cout<<fnamn<<" är ett fint namn"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Ja, så som du kan se, så fungerade det med cin>> på precis samma sätt, som när vi läste in heltal och doubles i tidigare exempel. En körning av programmet gav följande utskrift:

```
Vad heter du? Morgan
Morgan är ett fint namn
Tryck på en valfri tangent för att fortsätta....
```

Det finns dock ett problem med denna inläsning.

instring_fullname_1.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
string namn;
```

```
    cout<<"Vad heter du? Förnamn och efternamn: ";
    cin>>namn;
    cout<<namn<<" är ett fint namn"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

I det här programmet, så har jag döpt om variabeln fnamn till namn, eftersom jag tänkt mig att lagra uppgifter om både förnamn och efternamn i samma variabel.

Se nedan, hur det såg ut vid en provkörning.

```
Vad heter du? Förnamn och efternamn: Morgan Augustsson
Morgan är ett fint namn?
Tryck på en valfri tangent för att fortsätta...
```

Ja, trots att jag så mödosamt gjorde precis som programmet bad mig, skrev in både förnamn och efternamn, så tycktes mitt efternamn försvinna.

Det beror på att cin tolkar mellanslaget som att strängen är slut. För att lösa den uppgiften, så får man lov att använda en annan funktion. Det finns en funktion, som heter getline. Den ska ha två argument, cin och en strängvariabel. Man kan, om man så önskar, skicka med det tecken man vill att inläsningen ska brytas vid, t.ex. ett enterslag , '\n'. Jag visar även denna variant i koden, men har kommenterat bort den.

instring_fullname_2.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
string namn;
```

```
    cout<<"Vad heter du? Förnamn och efternamn: ";
    getline(cin,namn);
    //getline(cin,namn,'\n'); tom med enterslag
    cout<<namn<<" är ett fint namn"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En körning av programmet gav utskriften nedan. Glädjande nog, så kom hela namnet med den här gången.

```
Vad heter du? Förnamn och efternamn: Morgan Augustsson
Morgan Augustsson är ett fint namn
Tryck på en valfri tangent för att fortsätta...
```

Att jämföra två strängar är relativt enkelt. Man kan använda jämförelseoperatorn ==. Vi ska kika på det gamla kära inloggningsprogrammet. *Det är för övrigt så att man även kan använda operatorerna <, >, <= och >= för alfabetisk jämförelse av två strängar*

inloggning.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
    dos_console();
string password="superhemligt";
string usertry;

    cout<<"Lösenordet tack:";
    cin>>usertry;
```

```
        if(usertry == password)
                cout<<"Du är mycket välkommen!"<<endl;
        else
                cout<<"Nä,det gick inte så bra"<<endl;



    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Jag tror nog inte att någon närmare förklaring till koden behövs. Förmodligen, så hade du klarat av att skriva programmet själv vid det här laget. Några programkörningar kan du se här nedan.

```
Lösenordet tack:isglass

Nä,det gick inte så bra

Tryck på en valfri tangent för att fortsätta...

Lösenordet tack:superhemligt

Du är mycket välkommen!

Tryck på en valfri tangent för att fortsätta...
```

Man kan också arbeta med delsträngar. Det finns en funktion, substr för detta ändamål. Man bör i det sammanhanget tänka på att strängens första tecken har adressen 0.

del_string.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>


using namespace std;


int main()
{
dos_console();
```

```
string str="Morgan från Hjo";
string del_ett=str.substr(11);
string annan_del = str.substr(0,6);
cout<<"den ursprungliga strängen = "<<str<<endl;
cout<<"del_ett="<<del_ett<<endl;
cout<<"del två = "<<annan_del<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Som du kan se, så finns funktionen i två utförande. Den första varianten innebär att man skickar med ett heltalsargument. Man får då en ny sträng, som börjar med det tecknet, som man har angett som argument plus resten av strängen. I det här sammanhanget är det viktigt att man förstår, att första tecknet har adressen 0. Den andra varianten substr(start,antal tecken framåt) ska ha två argument. Dels tecknet man vill att den nya strängen ska börja på, samt antal tecken framåt. Skulle man av misstag vilja ha mer tecken framåt, än vad som finns, så händer inte något allvarligt, utan man får de tecken som finns.

Programmet ovan gav följande utskrift:

```
den ursprungliga strängen = Morgan från Hjo
del_ett= Hjo
del två = Morgan
Tryck på en valfri tangent för att fortsätta...
```

Det finns många strängfunktioner och nästa vi ska kika på är replace. Med hjälp av denna funktion, så kan man byta ut delar i en textsträng mot nya uttryck. Funktionen finns i ett flertal varianter, dvs. funktioner som tar olika många argument. Den som visas i exempelprogrammet nedan byter ut texten med start i positionen för Tidaholm och tills längden på Tidaholm (8) med texten Hjo.

I exemplet används ytterligare en strängfunktion, nämligen find. Med hjälp av denna kan man ta reda på om, och i så fall vart ett uttryck börjar i en sträng. Skulle det sökta inte finnas i strängen, så returneras string::npos. Sökningen börjar till vänster. Det finns en variant, som börjar sin sökning från höger, som heter rfind.

replace.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
```

```
#include<iodos.h>


using namespace std;


int main()
{
    dos_console();
string str="Morgan bor i Tidaholm världens trevligaste stad";
string target="Tidaholm";
string replacement="Hjo";


    cout<<str<<endl;
    /* Funktionen find ger som resultat ett tal, som talar om vart
       i strängen någonting hittas. Skulle det eftersökta inte
       finnas i strängen, så returneras string::npos. */
    int start = str.find(target);
    /* Funktionen replace finns med ett flertal olika argument.
       Den som används här nedan ersätter texten Tidaholm med Hjo
       med start, där Tidaholm börjar och tills
       längden på Tidaholm.*/
    str = str.replace(start,target.size(),replacement);
    cout<<str<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande trevliga utskrift:

```
Morgan bor i Tidaholm världens trevligaste stad

Morgan bor i Hjo världens trevligaste stad

Tryck på en valfri tangent för att fortsätta...
```

Att hitta e-postadressen i en text

Vi ska nu använda strängklassen till något mer nyttigt, innan vi fortsätter med utforskandet av CSträngar. Låt oss anta att vi har en text, som ser ut så här ”Om du har några frågor om produkterna,

garantier och priser, så kan du maila oss Fråga oss . Vi besvarar dina mail så fort som möjligt.”.

Texten är förstås hämtat från en webbsida. Nu skulle vi vilja ha tag på e-postadressen, för att t.ex. kunna spara ner den på fil.

Vad har vi för data att arbeta med?

Ja vi har förstås strängen som indata. Utdata blir den e-postadress, som extraheras ur texten. Vi behöver också veta startposition och slutposition för e-postadressen.

Vilka instruktioner är lämpliga?

Jo man skulle kunna börja med att lokalisera @. Därifrån skulle vi kunna backa, tecken för tecken, tills vi hittar en rimlig början på en e-postadress, mellanslag, citat eller kolon. Efter den operationen, så har vi en startpunkt för e-postadressen. Sedan går vi åt andra hållet, tecken för tecken, tills vi hittar ett rimlig slut för en e-postadress. På det viset får vi också en slutpunkt för e-postadressen. Med hjälp av startpunkten och slutpunkten, så kan vi ta ut delsträngen, som innehåller e-postadressen.

e-postadress.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>

using namespace std;

int main()
{
    dos_console();
string str="Om du har några frågor om produkterna, garantier ";
str += "och priser, så kan du maila oss ";
str += "<a href=\"mailto:kundinfo@enterprise.com\">Fråga oss </a>.";
str += " Vi besvarar dina mail så fort som möjligt.";
int startpunkt=0;
int slutpunkt=0;
string adress;
                startpunkt= str.find("@");
                if(startpunkt != string::npos)
```

```
                {
                slutpunkt=startpunkt;
                        for(int i=startpunkt;i>-1;i--)
                        {
                            if(str[i]==' ' ||
                                str[i]==':' ||
                                 str[i]=='"')
                            {
                                startpunkt=i+1;
                                break;
                            }
                        }
                        for(int i=slutpunkt;i<str.size();i++)
                        {
                            if(str[i]==' ' ||
                                str[i]==':' ||
                                    str[i]=='"')
                            {
                                slutpunkt=i;
                                break;
                            }
                        }
                }
       adress = str.substr(startpunkt,slutpunkt - startpunkt);
       cout<<adress<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

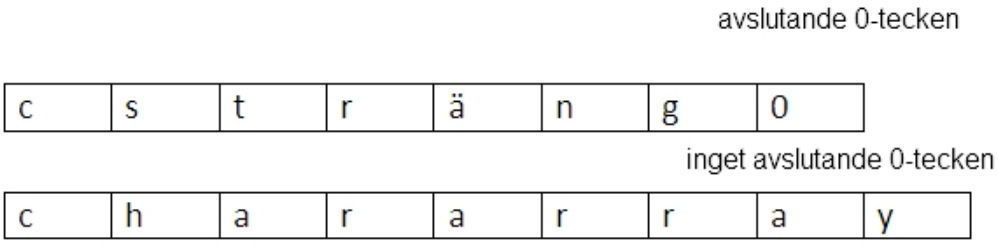
Programmet gav följande trevliga utskrift:

Tryck på en valfri tangent för att fortsätta...

Csträngar

I programmeringsspråket C, som är en föregångare till C++, så fanns inte string. Istället så använde man sig av någonting, som man kallade för csträngar. En csträng är en chararray med ett avslutande nolltecken. Observera att det måste vara ett nolltecken sist i arrayen. Finns inte det, så är det ingen sträng, utan en vanligt chararray.

Nu kanske du undrar varför du ska behöva lära dig det här, när det nu finns string? Det beror på att man förr eller senare hamnar i något bibliotek eller någon miljö, där csträngar förekommer. Det är inte alltid som man kan välja själv, tyvärr.



Bilden här ovan illustrerar skillnaden mellan en vanlig chararray och en csträng.

Det är lite besvärligare att arbeta med csträngar, än med string. Vi ska noga gå igenom de flesta moment med csträngar, som kan tänkas dyka upp i din programmeringsvardag. Vi ska väl ta och kika på hur man initierar och tilldelar en csträng.

cstring_1.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
    char namn_1[]="Morgan Augustsson";
```

```
    char namn_2[5]="Olof";


    cout<<"namn_1 = "<<namn_1<<endl;
    cout<<"namn_2 = "<<namn_2<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

namn_1 är en kompilatorskapad array. Skulle man försöka skriva ut sista tecknet, så kommer man att upptäcka att ett \0 tecken har lagts sist.

I namn_2, så har utrymmet beräknats efter antalet tecken i Olof plus ett. Hade utrymmet beräknats till antalet tecken i Olof enbart, så kunde problem ha uppstått. Det beror på att cout och alla de funktioner, som finns för csträngar, avgör vart strängen tar slut genom att se vart \0 tecknet ligger. Det är därför det är möjligt att skriva cout<<namn_1.

Programmet gav följande utskrift:

```
namn_1 = Morgan Augustsson
namn_2 = Olof
Tryck på en valfri tangent för att fortsätta...
```

Och visst, så kan man läsa in även csträngar från tangentbordet. Visserligen uppstår samma problem som med string, om man försöker skriva in flera ord, åtskiljda med mellanslag. Som tur är, så finns det medicin även mot detta. Först dock en vanlig enkel inläsning av ett ord.

cstring_2

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
    char fnamn[50];
    cout<<"Vad har du för tilltalsnamn:";
```

```
    cin>>fnamn;
    cout<<fnamn<<" är ett fint namn!"<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Ett bekymmer I sammanhanget är att man inte på förhand kan veta, hur långt namn, som kommer att matas in. Man kan bara gissa och anta. Jag antar att 50 tecken räcker till de allra flesta förnamn.

```
Programmet gav följande utskrift:

Vad har du för tilltalsnamn:Valdemar

Valdemar är ett fint namn!

Tryck på en valfri tangent för att fortsätta...
```

Så var det då bekymret med inmatning av flera ord till samma csträng. Se nedan!

ctring_3.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>


using namespace std;


int main()
{
dos_console();
    char namn[50];
    cout<<"Vad heter du? Förnamn och efternamn:";
    cin>>namn;
    cout<<namn<<" är ett fint namn!"<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift vid en testkörning:

```
Vad heter du? Förnamn och efternamn:Morgan Augustsson

Morgan är ett fint namn!

Tryck på en valfri tangent för att fortsätta...
```

Problemet är precis detsamma, som när vi läste in en string. cin tolkar mellanslaget, som ett slut på strängen. Vi måste i det här särskilda fallet använda en annan funktion för inläsning. Funktionen heter getline och tar två argument. Det första argumentet är den csträng, som ska fyllas med innehåll och det andra argumentet är ett heltal, som anger hur många tecken högst, som ska läsas in. Funktionen läser normalt en hel rad, men högst det antal tecken man har angivit som argument två. Funktionen lägger dessutom automatiskt in ett avslutande \0 tecken.

```
cstring_4.cpp
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
    char namn[50];
    cout<<"Vad heter du? Förnamn och efternamn:";
    cin.getline(namn,50);
    cout<<namn<<" är ett fint namn!"<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift vid en testkörning:

```
Vad heter du? Förnamn och efternamn:Morgan Augustsson

Morgan Augustsson är ett fint namn!

Tryck på en valfri tangent för att fortsätta...
```

Funktionen getline löste problemet. Vi kan göra program, som kan hantera både förnamn och efternamn på samma bräda.

Det finns mängder med funktioner för bearbetning av csträngar. Om man vill ta reda på, hur lång en sträng är, så kan man använda en funktion, som heter strlen. Den tar som argument, den sträng man vill veta längden på och returnerar längden i form av ett heltal. Man bör dock vara på det klara med att den längd, som returneras, är längden minus det avslutande \0 tecknet. Det kan ha sina komplikationer i olika sammanhang. Jag ska nu demonstrera att csträngar också är chararrayer. Om man anger sitt namn på tangentbordet och lagrar det i en chararray, så borde man, genom att iterera från slutet i arrayen till början med till exempel en for-loop, kunna få veta vad man heter baklänges.

cstring_5.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
    char namn[50];
    cout<<"Vad heter du? Förnamn och efternamn:";
    cin.getline(namn,50);
    cout<<namn<<"Ditt namn baklänges blir:"<<endl;
    int length = strlen(namn);
    for(int i=length-1;i > -1;i--)
        cout<<namn[i];
    cout<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Det finns kanske ett par saker i koden, som du blir lite förbryllad över. Till att börja med, så sker tilldelning av variabeln i i for-loopen till strängens längd minus 1. Tänk på att arrayer är

nollindexerade. Det innebär att sista adressen i en array är lika med dess längd -1. Villkoret i>-1 i for-loopen är till för att vi ska få en utskrift av tecknet på adress 0.

En testkörning av programmet gav följande utskrift:

```
Vad heter du? Förnamn och efternamn:Morgan Augustsson

Morgan Augustsson är ett fint namn! Och baklänges så blir det:

nosstsuguA nagroM

Tryck på en valfri tangent för att fortsätta...
```

Jag tror nästan att jag skulle kunna uttala mitt namn baklänges. Kan du?

Vi ska nu titta på ett annat litet problem med csträngar. Vi göra om inloggningsprogrammet, som vi gjorde tidigare, men nu ska vi använda csträngar istället för string. Jag har valt att använda cin för inläsning av användarens lösenord. Det går väl bra, eftersom lösenordet inte ska innehålla några mellanslag.

cstring_5.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
    dos_console();
char password[]="superhemligt";
char usertry[50];

    cout<<"Lösenordet tack:";
    cin>>usertry;
        if(usertry == password)
                cout<<"Du är mycket välkommen!"<<endl;
        else
                cout<<"Nä,det gick inte så bra"<<endl;
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Sedan testar vi:

```
Lösenordet tack:superhemligt
Nä,det gick inte så bra
Tryck på en valfri tangent för att fortsätta...
```

Va!? Jag skrev ju rätt lösenord, men jag lyckades ändå inte logga in. Det beror på att man visserligen kan skriva if(cstring_a == cstring_b), men det betyder inte det man skulle kunna tro. Vi ska längre fram i boken gå igenom en särskild variabeltyp, som hanterar primärminnesadresser, s.k. pekare. I C++, så är pekare och arrayer, så intimt sammanknippade att man kan säga, att de är samma sak. Uttrycket ovan betyder, om a och b refererar till samma variabel, så returneras sant. Nä, vi måste göra på ett helt annat sätt. Det finns en funktion i biblioteket cstring, som heter strcmp (stringcompare). Den tar två csträngar som argument och returnerar 0 om de båda är lika. Om första argumentet är mindre än det andra, så returneras ett negativt tal. Med mindre menas, att ordet kommer före i alfabetet. Kom ihåg, att char egentligen är ett heltal, ett index i en teckentabell. Om det andra argumentet är mindre än det första, så returneras ett positivt tal. Det här är återkommer vi till längre fram i boken, när vi ska sortera textsträngar. Vi skriver om programmet en aning. Det kan tänkas, att du måste inkludera cstring i ditt program, för att du ska få tillgång till strcmp.

cstring_6.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<cstring> //Denna inkludering kan behövas

using namespace std;

int main()
{
    dos_console();
char password[]="superhemligt";
char usertry[50];
```

```
    cout<<"Lösenordet tack:";
    cin>>usertry;
        if(strcmp(usertry,password)==0) //Nytt
                cout<<"Du är mycket välkommen!"<<endl;
        else
                cout<<"Nä,det gick inte så bra"<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Ja, då testar vi igen då?

```
Lösenordet tack:superhemligt
Du är mycket välkommen!
Tryck på en valfri tangent för att fortsätta...
```

Jo, den här gången lyckades jag att logga in. Det finns fler saker med csträngar, som skiljer från string. Om man har två Csträngar, a och b, så kan man inte göra en tilldelning från den ena till den andra med ett lika med tecken.

```
char a[]="Morgan";
char b[]="Pelle";
b=a;
```

Ovanstående ger helt enkelt ett kompileringsfel. Lösningen är funktionen strcpy (stringcopy). Den tar två argument, målsträngen och källan. Det finns en variant av funktionen, som heter strncpy. Den tar utöver de två argument, som strcpy tar ytterligare ett heltal som argument. Med det tredje argumentet anger man max tecken, som ska kopieras över till målsträngen. I funktionen strcpy görs ingen som helst kontroll av att det finns tillräckligt med plats i målsträngen.

Okey, då kikar vi på ett exempel.

cstring_7.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
```

```
#include<cstring>


using namespace std;


int main()
{
    dos_console();
char a[]="Morgan";
char b[9];
char c[3];
        strcpy(b,a);
        strncpy(c,b,2);
        c[2]='\0'; //Var tvungen att själv lägga in \0
        cout<<a<<" "<<b<<" "<<c<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

I exemplet ovan, så var jag tvungen att lägga in ett 0\ tecken sist i c. Det beror på att b är längre än c.

Följande utskrift gavs vid en provkörning:

```
Morgan Morgan Mo

Tryck på en valfri tangent för att fortsätta...
```

När vi lite tidigare tittade på string, så såg vi, att man kan sätta ihop strängar (konkatenera) med tecknet + . Så väl ordnat är det inte med Csträngar, utan man får i vanlig ordning använda någon särskild funktion till detta. Funktionen heter strcat och tar två Csträngar som argument, där den andra Cströngen läggs i slutet på den första. I den här funktionen sker ingen kontroll av att det finns tillräckligt med utrymme. Det gör det däremot i varianten strncat, som tar ytterligare ett argument, ett heltal, som man använder för att ange max antal tecken, som får läggas till. Vi ska väl i vanlig ordning kika på lite kod.

cstring_8.cpp

```
#include <cstdlib>
#include <iostream>
```

```
#include<iodos.h>
#include<cstring> //Denna inkludering kan behövas

using namespace std;

int main()
{
    dos_console();
    char a[40] = "Morgan";//Väl tilltagen array
    char b[]="Augustsson"; //kompilatorskapad array
    char c[20]="Datanörd"; //kompilatorskapad array
    cout<<a<<endl;
    cout<<b<<endl;
    strcat(a," ");//Lägger på ett mellanslag
    strcat(a,b);
    cout<<"a och b tillsammans så blir det: "<<a<<endl;
    strcat(a," ");//Lägger på ett mellanslag igen
    /*På nästa rad räknar jag dynamiskt ut hur
      mycket plats det finns kvar i a. Jag kommer att
      lägga in max det utrymmet. Det skulle kunna innebära
      att inte hela c får plats. Observera att strlen ger
      csträngens längs utom \0 tecknet. Det får man således
      beräkna med strlen + 1
    */
    strncat(a,c,40-strlen(c)+1);
    cout<<"a, b och c sammanslagna blir: "<<a<<endl;



    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande illustrative utskrift:

```
Morgan

Augustsson

a och b tillsammans så blir det: Morgan Augustsson

a, b och c sammanslagna blir: Morgan Augustsson Datanörd

Tryck på en valfri tangent för att fortsätta...
```

Att knäcka ett lösenord med slump och en Csträng

Vi har ju redan skapat ett inloggningsprogram. Kanske dags att skapa ett program att använda, om man inte kan lösenordet. Det finns ganska många etablerade lösenordsknäckaralgoritmer. En av de mer kända kallas för Brute Force och innebär att man plockar ihop tecken ungefär som i ett gammalt räkneverk. En startsekvens kan se ut så här, aa, ab, ba och bb. Vi ska testa Brute Force också, men vi ska börja med att slumpa fram en lösning. Vi ska för enkelhetens skull försöka oss på ett lösenord, som består av fyra tecken. När vi slumpar, så ska vi använda alla bokstäver från a till z både som gemener och versaler samt alla siffror.

Vad har vi för data att arbeta med?

Lämpligt är väl förstås att vi har en csträng, som innehåller alla bokstäver från a till z både som gemener och versaler samt alla siffror. Vi kan kalla den för tecken. Vi bör ha en csträng med plats för fyra tecken, som vi kan kalla för test. Vi behöver slumpa fram fyra stycken ingångar i teckenarrayen, för att fylla testarrayen. Vi skulle kunna använda en bool, som vi kallar för klar och som ställs till värdet false ända tills vi har hittat lösenordet. Intressant kunde det också vara att ha en variabel av heltalstyp, som håller reda på, hur många försök, som behövdes för att hitta lösenordet. Vi kan kalla den för times.

Vilka instruktioner är lämpliga?

Man skulle kunna tänka sig en while-loop, som inte avslutas förrän lösenordet har hittats. I whileloopen slumpas fyra ingångar i arrayen med tecken och placeras i testarrayen. Variabeln times räknas upp med ett. Sedan jämför vi testarrayen, som är en Csträng, med lösenordet. Om orden är lika, så avslutas whileloopen och ett positivt resultat skrivs ut på skärmen.

Vi kunde ju börja med att försöka slumpa fram några ord.

password_breaker_slump.cpp

```
#include <cstdlib>
#include <iostream>


using namespace std;


int main()
```

```
{
char tecken[]=
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";
char test[5];
int start=0;
    srand(time(0));

    while(start<5)
    {
        for(int i=0;i<4;i++)
        {
            int plats=rand()%strlen(tecken);
            test[i]=tecken[plats];
            /*För att vara säker på att CSträngen avslutas
              med ett 0-tecken, så läggs ett sådant in strax
              efter platsen där tilldelning har skett. Det hade
              räckt med att lägga det sist efter avslutad
              for-loop, men det spelar nog inte någon större
              roll.*/
            test[i+1]='\0';
        }
    cout<<test<<endl;
    start++;
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En testkörning av programmet gav följande utskrift:

```
rq9S
zqJg
ISm7
```

```
SMvu

W3zh

Tryck på en valfri tangent för att fortsätta...
```

Fem stycken prydligt obegripliga ord. Precis som lösenord helst ska vara. Då återstår väl egentligen bara att göra färdigt programmet. Observera att jag har använt iodos i det färdiga programmet. Det som lagts till har fet stil.

password_breaker_slump.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;


int main()
{
    dos_console();
char tecken[]=
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";
char password[]="lurA";
bool klar=false;
char test[5];
int times=0;
    srand(time(0));


    while(!klar)
    {
    times++;
        for(int i=0;i<4;i++)
        {
            int plats=rand()%strlen(tecken);
            test[i]=tecken[plats];
            test[i+1]='\0';
```

```
        }
        if(strcmp(password,test)==0)
        {
            cout<<"Lösenordet ["<<test<<
            "] hittades efter "<<times<<" försök."<<endl;
            klar = true;
        }
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En testkörning av programmet gav följande utskrift:

```
Lösenordet [lurA] hittades efter 2629971 försök.

Tryck på en valfri tangent för att fortsätta...
```

Som du kan se, så fordrades det ganska många försök innan lösenordet hittades. Pröva gärna att byta lösenord. Testa också mycket gärna med ett längre lösenord.

Att knäcka ett lösenord med Brute Force

Att käcka ett lösenord med Brute Force innebär att man kombinerar de tecken, som kan ingå i lösningen på ett effektivt sätt. Man prövar helt enkelt alla tänkbara kombinationer. Jag klarade själv en gång i tiden av att öppna ett hänglås med tresiffrig kombination. Det var inget olagligt i det, utan jag hade helt enkelt glömt bort den öppnande kombinationen. Jag började helt enkelt med 0,0,0. Fortsatte sedan med 0,0,1 osv. Har man tur, så ligger lösningen någonstans i början. Har man otur, så ligger den sist, eller nästan sist.

I ett dataprogram, så skulle man kunna tänka sig ett flertal lösningar. En elegant modell arbetar med rekursion, men det är ganska komplicerat och faller därför utanför ramarna för en nybörjarbok i programmering. Jag har valt, att använda nästlade for-loopar. Att nästla en for-loop innebär att man lägger en for-loop inuti en annan for-loop. Vi ska kanske börja med att kika på ett litet exempel på en nästlad for-loop. *Du kan se ett exempel på detta även i avsnittet om iteration med for, där vi gjorde en enkel multiplikationstabell.*

nastlad_forloop.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;
```

```
int main()
{
    for(int i=0;i<3;i++)
    {
        for(int j=0;j<3;j++)
        {
            cout<<"i="<<i<<" j="<<j<<endl;
        }
    cout<<"********************"<<endl;
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Om du studerar utskriften, så kan du se, hur den inre for-loopen körs en gång för varje steg i den yttre for-loopen.

```
i=0 j=0
i=0 j=1
i=0 j=2
********************
i=1 j=0
i=1 j=1
i=1 j=2
********************
i=2 j=0
i=2 j=1
i=2 j=2
********************
Tryck på en valfri tangent för att fortsätta...
```

Du kan till och med se, att alla kombinationer av 0,1 och 2 finns med i utskriften. Nu behöver man inte alls begränsa sig till en nästlad for-loop. Man kan i princip nästla hur många som helst. Om förutsättningar, när vi nu ska knäcka lösenordet med Brute Force är desamma, som när vi gjorde det med slump, så ska vi använda fyra nästlade for-loopar, för att få testord, som är fyra tecken långa. Den enda svårigheten med den här lösningen är att få for-looparna att stanna av, när man har hittat lösenordet. Man kan avbryta en iteration med uttrycker break, men det avbryter enbart den omslutande iterationen. De for-loopar, som ligger runtom, fortsätter som vanligt. Jag har löst det i mitt exempelprogram, genom att helt enkelt avsluta programmet, när lösningen har hittats.

brute_force.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;

int main()
{
    dos_console();
char tecken[]=
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";



char password[]="lurA";
bool klar=false;
char test[5];
int times=0;
    srand(time(0));

    while(!klar)
    {

        for(int i0=0;i0<strlen(tecken);i0++)
        {
            for(int i1=0;i1<strlen(tecken);i1++)
```

```
            {
                for(int i2=0;i2<strlen(tecken);i2++)
                {
                    for(int i3=0;i3<strlen(tecken);i3++)
                    {
                    times++;
                    test[0]=tecken[i0];
                    test[1]=tecken[i1];
                    test[2]=tecken[i2];
                    test[3]=tecken[i3];
                    test[4]='\0';
                        if(strcmp(password,test)==0)
                        {
                        cout<<"Lösenordet ["<<test<<
                        "] hittades efter "<<times<<
                        " försök."<<endl;
                        system("PAUSE");
                        return EXIT_SUCCESS;

                        }
                    }
                }
            }

    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En provkörning av programmet gav följande utskrift:

```
Lösenordet [lurA] hittades efter 2699569 försök.
```

```
Tryck på en valfri tangent för att fortsätta...
```

I all sin enkelhet, så fungerade programmet och lösenordet knäcktes. Pröva gärna att bygga ut programmet, så att du kan lösa längre och mer komplicerade lösenord. Om man försöker lösa långa lösenord, så kan man få vänta länge, ja till och med väldigt länge. En annan sak som är lite besvärlig med den här konstruktionen är förstås, att man normalt sett inte kan veta, hur många tecken det finns i lösenordet.

Funktioner

Många gånger upptäcker man i sitt programmerande att man utför samma uppgift gång på gång. Det kan t.ex. handla om att välja det lägsta värdet mellan två variabler, eller att plocka ut det lägsta värdet ur en array osv. Då skulle man vilja lösa problemet en gång och sedan på något sätt återanvända lösningen. En funktion skulle kunna vara lösningen på problemet. Jag vågar påstå att funktioner är ett väldigt viktigt verktyg i programmerarens verktygslåga.

En funktion, skulle kunna beskrivas, som ett stycke kod under ett namn, som man kan anropa via namnet och eventuellt ett eller flera argument. Vi har tidigare i boken använt funktioner, som någon annan har skapat, t.ex. strlen med en cssträng som argument.

En funktion består av ett funktionshuvud och en kropp enligt nedanstående schema:

returtyp	namn	Argumentlista (formella parametrar)

```
{
        Funktionens kropp omsluts av krullparanteser.

        Här i kroppen återfinns deklaration, tilldelningar

        Och algoritmer.

        I funktioner som returnerar något, så kan det

        finnas en eller flera returnsatser.  Sådana kan

        finnas även i funktioner, som inte returnerar något, men

        det är inte så vanligt. En returnsats får exekveringen av

        funktionen att upphöra.

}
```

Det skulle kunna se ut på följande sätt:

```
int max(int a,int b) // En funktion, som returnerar en int, heter max och tar två heltal som argument

int storst = max(10,8); //Så här skulle det kunna se ut, när man anropar funktionen

void skrivTecken(char tecken, int antal)//Funktionen returnerar ingenting, heter skrivTecken och tar en char och ett heltal, som argument
```

skrivTecken('*',10);// Exempel på användning

Man skulle grovt sett kunna dela in funktionerna i två huvudgrupper, funktioner, som returnerar något och funktioner som gör något. Vi ska börja med att kika på några exempel, där vi skapar och använder funktioner, som gör något, men som inte returnerar något.

Funktioner, som inte returnerar något (void)

Okey vi kanske skulle titta på ett exempel:

funktioner_1

```
#include <cstdlib>
#include <iostream>

using namespace std;

void skrivTecken(char tecken,int antal)
{
int start = 0;
    while(start++ < antal)
        cout<<tecken;
    cout<<endl;
}

int main()
{
    skrivTecken('*',5);
    skrivTecken('+',10);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Som du kan se, så har vi här en funktion, som heter skrivTecken längst upp i koden. Den returnerar ingenting, vilket innebär att vi inte får någonting tillbaka, när den anropas. Void betyder, som bekant tomt. Den tar en char och ett heltal som argument.

Nu kanske du ser att main också är en funktion. Vart anropas main? Jo, main anropas av operativsystemet, som när programmet laddas in i primärminnet, letar efter en punkt att starta exekveringen av programmet i. Skulle vi till exempel döpa om main till Main, så får vi inget program.

Programmet avsluts i och med att EXIT_SUCCESS returneras till operativsystemet. Det betyder att operativsystemet informeras om att programmet är färdigt. EXIT_SUCCESS är en konstant till heltal.

I main, så sker ett anrop till skrivTecken med argumenten ’*’ och 5. Det får till följd att exekveringen av programmet flyttas från anropsstället till funktionen. ’*’ och 5 är så kallade aktuella parametrar. De formella parametrarna får värdet ‘*’ och 5, dvs. tecken får värdet ’*’ och antal värdet 5. Dessa värden gäller inom funktionen skrivTecken. När funktionen har exekverat färdigt, så försvinner dessa värden och exekvering återgår till raden under anropsstället i main. Där sker så ett nytt anrop till skrivTecken, den här gången med de aktuella parametrarna ’+’ och 10. När funktionen exekverat färdigt, så återgår egentligen exekveringen till raden under anropsstället och programmet körs färdigt.

En körning av programmet gav följande utskrift:

```
*****

++++++++++
Tryck på en valfri tangent för att fortsätta...
```

Man kan fundera lite över placeringen av funktionen skrivTecken. Skulle vi flytta den och lägga den under mainfunktionen, vilken vore det bästa för läsbarhetens skull, så får vi problem, när vi kompilerar. Pröva att ändra i koden enligt nedan:

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    skrivTecken('*',5);
    skrivTecken('+',10);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivTecken(char tecken,int antal)
```

```
{
int start = 0;
    while(start++ < antal)
        cout<<tecken;
    cout<<endl;
}
```

Min kompilator i DevC++ klagade och gav mig följande felmeddelande: 8 E:\skolan\2010-2011\programmering_b\cpp_boken\funktioner\main.cpp `skrivTecken' undeclared (first use this function)

Oj då! Vad var det som gick fel?

Jo, kompilatorn hävdar att funktionen skrivTecken inte finns. Men tittar jag i min kod, så vågar jag påstå att kompilatorn ljuger, funktionen finns där.

För att förstå problemet, så måste man också förstå kompilatorns arbetssätt. Den börjar längst upp i filen och går sedan nedåt, rad för rad. När den kommer till första anropsstället i mainfunktionen, så ska den hoppa till funktionen skrivTecken, men den tittar bara uppåt, eller mer korrekt på den kod, som den redan har bearbetat. Den kod, som inte har bearbetats, eller ligger under kompilerad kod existerar inte. Det var därför som det fungerade, när vi hade funktionen överst.

Funktionsparametrar

Men ska man placera alla sina egenskapade funktioner överst i sina program, så kommer det att bli ganska rörigt, åtminstone om man gör lite större program. Den bästa lösningen vore förstås att placera funktionerna i en fil för sig. Det kommer vi också att göra lite längre fram i boken, men låt oss först lösa problemet utan, utan att använd fler filer.

Lösningen är att placera en funktionsprototyp ovanför main. Sedan kan man placera själva funktionen under main. En funktionsprototyp är funktionshuvudet följt av ett semikolon, men ingen funktionskropp.

void skrivTecken(char tecken,int antal); // En prototyp

Det finns ganska mycket att säga om funktionsprototyper, bland annat, så måste man inte ge parametrarna några namn.

void skrivTecken(char,int);//fungerar också

Detta är någonting, som du kommer att råka på, förr eller senare i ditt programmerande. Om du vill använda en funktion, som finns i något bibliotek, som du har hämtat på nätet, eller i din kompilatorinstallation, så hittar du funktionsprototyperna i en headerfil. Det är inte alls säkert att du kommer åt källkoden till själva funktionerna. Då är funktionsprototyperna kanske den enda information, som är tillgänglig, för hur funktionen är tänkt att användas. Skulle då programmeraren, som skapat funktionen utelämnat namn på parametrarna, så är det mycket svårare att förstå funktionen. Skulle den dessutom ha ett mer eller mindre obegripligt namn, så blir det ännu svårare.

Därför vill jag starkt rekommendera, att du ger parametrarna beskrivande namn och i möjligaste mån namnger funktionerna på samma sätt.

Man kan också ge de formella parametrarna defaultvärden i funktionsprototyper. Mer om detta längre fram. Okey då lägger vi dit en prototyp i vår kod. Kompilatorn kommer att läsa in prototypen och förstår på det viset att någon annan stans ligger själva funktionen. Skulle man råka utelämna källkoden till funktionen, så kommer man att få ett länkningsfel vid kompileringen.

funktioner_2.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;
void skrivTecken(char tecken,int antal);

int main()
{
    skrivTecken('*',5);
    skrivTecken('+',10);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivTecken(char tecken,int antal)
{
int start = 0;
    while(start++ < antal)
        cout<<tecken;
    cout<<endl;
}
```

Och nu fungerade det betydligt bättre. I nästa exempel, så ser vi ett exempel på en funktion, som saknar argument i parameterlistan. Då har man helt enkelt en tom parentes. Det är fullt tillåtet att skriva void i parentesen, men inte nödvändigt. När man anropar en sådan funktion, så använder man också en tom parentes.

funktioner_3.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

void skrik();

int main()
{
int start = 0;
const int MAX=10;

    while(start++<MAX)
            skrik();//Anrop av parameterlös funktione
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrik()
{
    cout<<'\7';
}
```

I funktionen skriv, så skrivs tecknet ’\7’ ut. Backslashen är ett flykttecken. Trots att funktionen skriver ut någonting, så syns ingenting på skärmen, men däremot så piper det i datorn. Man skulle kunna säga att vi skriver ut ett ljud till högtalaren istället för ett tecken på skärmen. Sju är asciikoden för pc-bipet.

Du observerade väl också att i anropet användes en tom parentes skrik().

Defaultparametrar

Men nu kanske man skulle vilja skicka med ett argument, som anger hur många pip man vill ha, samtidigt som man ibland skulle vilja göra anropen utan argument. Då kan man använda sig av ett defaultvärde och lämpligaste stället att ange ett defaultvärde på är i funktionsprototypen. Vi skriver om programmet lite.

funktioner_4.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;
void skrik(int antal=5);

int main()
{
    skrik();
    skrik(2);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrik(int antal)
{
int start=0;
    while(start++<antal)
        cout<<'\7';
}
```

Du kan se att funktionsprototypen för skrik ser lite annorlunda ut. I argumentlistan, så har heltalsvariabeln antal fått ett värde, eller rättare sagt, den kommer att få det, om man vid anropet inte skickar med något argument. Nu kan man omväxlande anropa funktionen med och utan argument, vilken framgår av programmet.

Skulle det vara på det viset, att man har fler argument, så måste alla argument, som följer ett argument med defaultvärde också ha defaultvärde.

Vi kikar i vanlig ordning på ett exempel på detta.

funktioner_5.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;
```

```
/*Observera att andra argumentet i parameterlista
  måste ha ett defaultargument, eftersom det första
  har det.*/
void skrivTecken(char tecken='*',int antal=5);

int main()
{
    skrivTecken();
    skrivTecken('x');
    /* Nedanstående funktionsanrop fungerar, men
       det blir inte riktigt vad man skulle kunna
       tro. Eftersom char också är heltal, så
       kommer 5 att översättas till ett tecken.
       Hade det inte varit möjligt att konvertera
       argumentet, så hade anropet varit felaktigt.*/
    skrivTecken(5);
    skrivTecken('W',7);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivTecken(char tecken,int antal)
{
int start=0;
    while(start++<antal)
        cout<<tecken;
    cout<<endl;
}
```

Programmet ger följande utskrift:

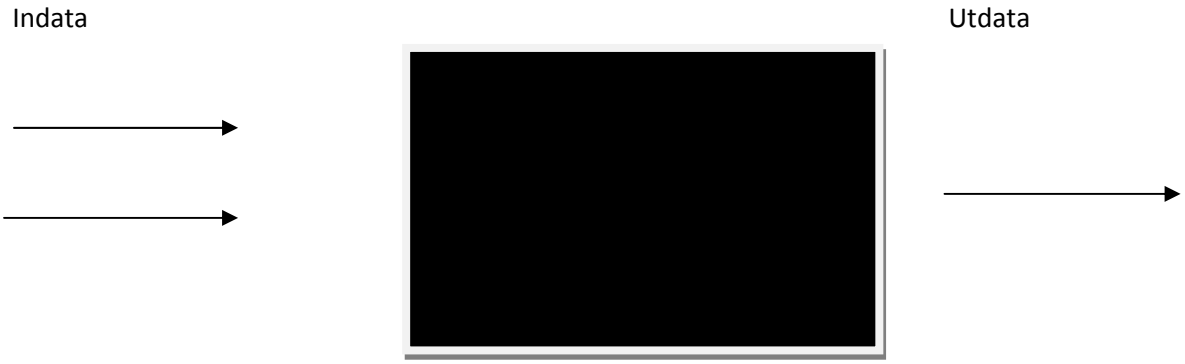
```
*****
```

```
xxxxx

♣♣♣♣♣

WWWWWWW

Tryck på en valfri tangent för att fortsätta...
```

Funktioner som returnerar någonting

Som vi har sett, så har de funktioner vi har skapat så här långt haft returvärdet void, vilket innebär att de inte har returnerat något. När det gäller funktioner med något annat returvärde än void, vilket som helst, så får man tillbaka någon form av värde vid ett anrop, t.ex. ett heltal, int m = Max(3,9).

Man brukar i programmeringslitteraturen beskriva en funktion, som en svart låda, från användarens synpunkt. Användare i det här fallet är en programmerare, som använder funktion. Man kan stoppa in något i låda och få någonting tillbaka. Vad som händer i koden är dolt och egentligen ointressant för användaren. Det kan till och med vara så, att skaparen av funktionen hittar en bättre lösning av problemet och därför skriver om koden i funktionen. Du som användare kan fortfarande skicka in samma typ/typer av argument och förvänta dig att få tillbaka ett processat och korrekt värde.



En funktion, som returnerar et värde gör det med uttrycker return värde. Vi ska skapa en funktion, som tar två heltal som argument och som returnerar det största av de två talen. Skulle talen vara lika stora, så spelar det förstås ingen roll vilket av talen som returneras. Jag vill starkt rekommendera att du ger dina funktioner beskrivande namn. Låt oss anta att du kallade den beskrivna funktionen för min! Det skulle säker upplevas, som frustrerande och total obegripligt av den som ska använda funktionen. Det är ganska så bra att man börjar med prototypen. Den är enkel att konstruera utifrån de krav vi har på den.

```
int Max(int a,int b);
```

Prototypen ovan kan nog inte missförstås av en användare. Nu ska vi skriva själva funktionen och då är det upp till oss att skriva kod, som löser uppgiften, som utlovas i prototypen. Ett förslag är att vi jämför a och b. Skulle a vara större än b, så returnera vi a. Ett returnuttryck får exekveringen att upphöra, så därför behövs inte något else, utan man kan helt enkelt returnera b. Om du för läsbarhetens skulle vill lägga dit en else, så står det dig naturligtvis fritt. Mitt förslag till funktion, ser ut så här:

```
int Max(int a,int b)
```

```
{
          ff(a>b)
          return a;
          // Här skulle man kunna ha en else, men det behövs inte
    return b;
}
```

Ja, då var det väl dags att pröva, om det fungerar som det ska.

funktioner_6.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

int Max(int a,int b);
int main()
{
int tal_1=11;
int tal_2=8;
int tal_3=8;
int tal_4=14;
        int greatest = Max(tal_1,tal_2);
        cout<<"greatest = "<<greatest<<endl;

        greatest = Max(Max(tal_2,tal_4),greatest);
        cout<<"greatest = "<<greatest<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Uttrycket, greatest = Max(Max(tal_2,tal_4), greatest), visar att returvärdet från ett funktionsanrop kan skickas vidare i sin tur som argument till en funktion. Vad som händer är att första argumentet i det här fallet, Max(tal_2,tal_4) körs innan det yttre funktionsanropet utförs. Man kan i princip nästla, hur många funktionsanrop som helst, men det är klart, någonstans så blir det rörigt och svårt att förstå koden.

En körning av programmet gav följande utskrift:

```
greatest = 11

greatest = 14

Tryck på en valfri tangent för att fortsätta...
```

Du kan väl på egen hand skapa en funktion, som tar två heltal som argument och som returnerar det lägsta talet.

Generiska funktioner

Uttrycket generiska funktioner kan översättas till generella funktioner. Låt oss anta, att man i ett program behöver anropa en funktion, som heter Max och som returnerar det högsta värdet av två inskickade argument. Ibland, så skickar man två heltal, ibland två decimaltal, för att emellanåt skicka två variabler av typen char. Hur gör man då? Jo, en lösning är förstås att skapa en funktion för vardera datatypen. En annan lösning är att använda en generell funktion, eller funktionsmall, som det också kallas för. Jag visar först ett exempel, innan jag berättar, hur det fungerar.

mall_funktion.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
template<class T>
T Max(T,T);
int main()
{
    cout<<Max('a','b')<<endl;
    cout<<Max(5,9)<<endl;
    cout<<Max(45.2,55.3)<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
template<class T>
```

```
T Max(T a,T b)
{
    if(a>b)
        return a;
    return b;
}
```

Om vi börjar med prototypen, så ser den ut så här:

```
template<class T>
T Max(T,T);
```

Template betyder mall på svenska. Man skapar på det här viset en allmän datatyp, som kallas för T. Du kan se att funktionen har returtypen T och tar dessutom två argument av typen T. Om vi raskt hoppar över till funktionsdefinitionen:

```
template<class T>
T Max(T a,T b)
{
    if(a>b)
        return a;
    return b;
}
```

Så ser vi att den har precis samma konstruktion, förutom att den har en funktionskropp förstås. När koden här ovan kompileras, så ser kompilatorn, att det sker tre funktionsanrop till Max. I det första fallet, så skickas två tecken som argument. Då skapar kompilatorn en funktion, som tar två tecken som argument och som returnerar ett tecken. Samma sak sker sedan för vart och ett av anropen, eftersom de har olika typer av argument. När kompileringen är slutförd, så har alltså programmet tre funktioner, en för vardera typen av argument.

Det här är förstås ett kraftfullt verktyg, om man behöver använda funktioner, som ska kunna användas för många typer av argument och returnera många olika typer av data.

Arrayer som argument till funktioner

Arrayer är intimt förknippade med en annan variabeltyp, som kommer att behandlas längre fram i boken, nämligen pekare. Men i det här avsnittet ska vi skicka iväg arrayer som argument till funktioner. I parameter listan, så skriver man enkelt en datatyp, ett namn och hakparenteser, t.ex. int tal[]. Man bör också som argument till funktionen skicka med antalet element i arrayen. Man kan alltså inte räkna ut antalet element på ett dynamiskt sätt i funktionen. För att föregå det som ska komma angående pekare, så skickar man egentligen inte en array som argument, utan en pekare. Det finns ett undantag från regeln om att man måste skicka med antalet element i en array, man kan

skicka med en cssträng, utan att samtidigt skicka med en uppgift om antalet element i strängen. Man kan istället använda funktionen strlen, eller så måttar man strängens slut efter '\0' – tecknet.

Vi ska i nästa exempel se en funktion, som skriver ut en heltalsarray på ett smakfullt sätt.

funktioner_7.cpp

```
#include <iostream>

using namespace std;

void skrivArray(int tal[],int antal);
int main()
{
    int numbers[]={8,23,11,45,19,2};//Kompilatorskapad array
int storlek = sizeof(numbers)/sizeof(int);//Beräknar antalet element

    skrivArray(numbers,storlek);//Anropar funktionen skrivArray
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivArray(int tal[],int antal)
{
    for(int i=0;i<antal;i++)
        cout<<"array["<<i<<"]="<<tal[i]<<endl;
}
```

Programmet ger följande utskrift:

```
array[0]=8
array[1]=23
array[2]=11
array[3]=45
array[4]=19
array[5]=2
```

```
Tryck på en valfri tangent för att fortsätta...
```

Det är ofta man har stor nytta av arrayer i lösningen av diverse problem. Vi kan föreställa oss, att vi har ett antal mätvärden, t.ex. ett antal provresultat från ett matteprov. Visst vore det väl smidigt, om vi hade en funktion, som tog provresultaten som argument och returnerade provresultatens medelvärde. Man skulle kunna komplettera det med funktion, som returnerade det bästa respektive det sämsta resultatet. En av finesserna med funktioner är förstås att de kan återanvändas. Med det menar jag inte bara, att en funktion kan anropas flera gånger i ett program, utan till och med i flera program. För att åstadkomma det, så borde man lägga funktionerna i en särskild fil, som man kan inkludera till alla de program, där de behövs.

Att skapa algoritmer metodik (2)– exempel

Ja, då ska vi ge oss i kast med några sköna utmaningar i programmeringens konst. Jag tänkte att vi skulle göra några krypterings och dekrypteringsövningar. Lite längre fram i det här avsnittet, så ska vi förvandla vanliga heltal till binärsträngar. Det innebär att vi måste fördjupa oss lite i det binära (tvåvärda) talsystemet. Vi börjar väl med kryptering . Självklart, så ska vi nu försöka använda funktioner i lösningarna.

Kryptering

Ibland vill man vara lite hemlig. Man skulle vilja skicka ett meddelande till en kompis. Eftersom ingen annan har med att göra, vad meddelandet innehåller, så skulle man vilja förvanska texten, så att det blir oläsligt för alla utom kompisen.

Tekniken går ut på, att man på något sätt kastar om bokstäverna enligt ett mönster som kompisen känner till, så att han/hon kan kasta tillbaka bokstäverna till ett läsbart meddelande.

Vad har vi för data att arbeta med?

Ja, först och främst, så har vi ett meddelande, som kan lagras i en variabel av typen string. Programmets utdata blir en sträng med det förvanskade meddelandet. Vi ska utnyttja förhållandet att strängar består av tecken. Tecken är av datatypen char och är egentligen heltal. Dessa kan man utföra mattematiska operationer på.

Vilka instruktioner är lämpliga?

Ja, man skulle kunna initierera strängen med det hemliga meddelandet, så att man kommer åt dess enskilda tecken. Dessa tecken, som är av datatypen char, kan vi göra om till heltal. Dessa heltal kan vi t.ex. minska värdet på med 12. De nya värdena kan vi åter göra om till tecken. Viktigt att tänka på i det här fallet är, att vi inte kan utföra någon operation på heltalen, som inte går att reparera. Man skulle löpa den risken, om man använde division på heltalen.

kryptering.cpp

```
#include <cstdlib>

#include <iostream>
```

```
#include<string>
#include<iodos.h>
using namespace std;
string krypt(string message);
int main()
{
dos_console();
string meddelande="Våra allierade har siktats i skogens "+
                  string("utkanter. \nDet är viktigt att vi ")+
                  string("inte röjer vår identitet.\nLösenordet ")+
                  string("är lingonben.");
    cout<<"Det okrypterade meddelandet:"<<endl;
    cout<<meddelande<<endl;
    string secret = krypt(meddelande);

    cout<<"Det hemliga meddelandet:"<<endl;
    cout<<secret<<endl;
   system("PAUSE");
    return EXIT_SUCCESS;
}
string krypt(string message)
{
string retval="";
    for(int i=0;i<message.size();i++)
    {
/*På nästa rad hämtas ett enskilt tecken ut från strängen
  med hjälp av arrayindexering. Eftersom det vi får tag på, ett
  tecken är av typen char, så kan vi göra om den till ett heltal
   med så kallad casting.*/
       int tecken_tal = message[i];
       tecken_tal-=12; //Lite matte på talet.
```

```
    /*På nästa rad, gör vi om heltalet till en char, som
      konkateneras med strängen retval, som ska returneras från
      från funktionen, när den krypterade strängen är klar*/

          retval+=(char)tecken_tal;

    }

    return retval;

}
```

Det finns förstås ett par saker i koden, som jag måste kommentera. Du kanske funderar över, vad,

```
string meddelande="Våra allierade har siktats i skogens "+

                  string("utkanter. \nDet är viktigt att vi ")+
```

osv. betyder?

Jo här konkateneras (klistras) flera strängar ihop till en. I det här fallet blir resultatet det meddelande, som ska krypteras.

Tecknet \n är en så kallad flyktfrekvens, som innebär att vi får en radbrytning i strängen.

Programmet gav följande utskrift:

```
Det okrypterade meddelandet:

Våra allierade har siktats i skogens utkanter.

Det är viktigt att vi inte röjer vår identitet.

Lösenordet är lingonben.

Det hemliga meddelandet:

JÙfU¶U``]YfUXY¶\Uf¶g]_hUhg¶]¶g_c[Ybg¶ih_UbhYf"¶þ8Yh¶Øf¶j]_h][h¶Uhh¶j]¶]bhY¶fê^Yf

¶jÙf¶]XYbh]hYh"þ@êgYbcfXYh¶Øf¶`]b[cbVYb"

Tryck på en valfri tangent för att fortsätta...
```

Huijanemej, det blir inte lätt att knäcka det här krypterade meddelandet.

Dekryptering

Jo, nu måste ju kompisen kunna låsa upp krypteringen och få ett läsbart meddelande. Vi kunde ju ha skickat det krypterade meddelandet med e-post, eller sparat meddelande i en fil. Jag har ju inte gått igenom filhantering (läsning/skrivning) ännu, så det får vänta tills dess.

Vad har vi för data att arbeta med?

Jo, vi har förstås den hemliga textsträngen. Precis, som när vi krypterade, så kan vi hämta ut tecken från strängen. Eftersom dessa är av typen char, så kan vi göra om dessa till tal. Talens värden ökar vi

med 12. Sedan görs talen om till tecken med hjälp av typkonvertering innan de konkateneras med strängen, som håller den dekrypterade strängen.

Vilka instruktioner är lämpliga?

Faktum är att vi gör på precis samma sätt, som när vi krypterade strängen, med ett undantag. Vi ökar talens värde med 12 istället för att minska med 12. Jag har för enkelhetens skull valt att lägga in funktionen för dekryptering i samma program, som krypteringen.

dekryptering.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>
using namespace std;
string krypt(string message);
string dekrypt(string message);
int main()
{
dos_console();
string meddelande="Våra allierade har siktats i skogens "+
                  string("utkanter. \nDet är viktigt att vi ")+
                  string("inte röjer vår identitet.\nLösenordet ")+
                  string("är lingonben.");
string secret = krypt(meddelande);
    cout<<"Det hemliga meddelandet:"<<endl;
    cout<<secret<<endl;
string dekrypterat=dekrypt(secret);
    cout<<"Det dekrypterade meddelandet:"<<endl;
    cout<<dekrypterat<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
string krypt(string message)
{
   /*Som tidigare */
}

string dekrypt(string message)
{
    string retval="";
    for(int i=0;i<message.size();i++)
    {
        int tecken_tal = message[i];
        tecken_tal+=12; //återställer det ursprungliga värdet.
        retval+=(char)tecken_tal;
    }
    return retval;
}
```

Programmet gav följande sköna besked:

```
Det hemliga meddelandet:
JÙfU¶U``]YfUXY¶\Uf¶g]_hUhg¶]¶g_c[Ybg¶ih_UbhYf"¶þ8Yh¶Øf¶j]_h][h¶Uhh¶j]¶]bhY¶fê^Yf
¶jÙf¶]XYbh]hYh"þ@êgYbcfXYh¶Øf¶`]b[cbVYb"
Det dekrypterade meddelandet:
Våra allierade har siktats i skogens utkanter.
Det är viktigt att vi inte röjer vår identitet.
Lösenordet är lingonben.
Tryck på en valfri tangent för att fortsätta...
```

Det binära talsystemet

Jag hade tänkt, att vi inte skulle nöja oss med att förvanska meddelandet genom att utföra någon operation på bokstäverna (tecknen) i strängen. Vi ska dessutom göra om de förvanskade bokstäverna till binärsträngar. För att det ska kännas meningsfullt, så måste jag berätta lite om, hur det binära talsystemet fungerar.

Jag har ju redan avslöjat att processorn är ganska korkad. Den kan endast utföra väldigt enkla operationer, såsom att addera, subtrahera, dividera och multiplicera två tal. Den kan också jämföra värden och se att 5 är större än 2.

Det språk som datorn använder kallas det binära (tvåvärda) språket. Att datorns språk är binärt beror egentligen på dess konstruktion. Man kan enkelt beskriva det som att processorn består av en massa strömbrytare. En strömbrytare har två lägen, på eller av. Siffran 1, betyder strömbrytare på och 0 betyder strömbrytare av.

De allra första persondatorer, som släpptes på marknaden saknade skärm. Man programmerade dessa genom att ställa om brytare för hand. Var man duktig, så kunde man faktiskt skapa enklare program på det viset. När programmet kördes, så blinkade det i ett antal lampor.

En brytare, eller enhet, kallas för en bit. En bit kan ha värdet 1 eller 0. Man brukar behandla dessa i grupper om 8. En sådan grupp brukar man kalla för en byte.

Om vi tänker oss att vi har en byte, dvs. åtta bitar, så skulle man kunna beskriva det med en bild, som ser ut så här:

1	1	1	1	1	1	1	1
128	64	32	16	8	4	2	1
summa	255						

I det binära talsystemet, så är det bitarnas position, som avgör dess värde. Man läser från höger till vänster. Om biten längst till höger är en etta, så har den värdet 1. Om andra biten från höger är en etta, så har den värdet två. Sedan ökar värdet för varje position med det dubbla värdet från den föregående positionen. Vi får en serie, 1, 2, 4, 6, 8, 16, 32, 64, 128. Har man ännu fler bitar att jobba med, så fortsätter ökningen av bitarnas värden på samma sätt. På bilden ovan, så har vi åtta bitar (en byte), där samtliga bitar har värdet ett. Om man adderar samtliga värden i rad två, så får man summan 255. Det innebär, att det högsta värde man kan lagra i en byte är 255. En datatyp, som jag har använt väldigt ofta i den här boken är int. Storleken på den kan variera, men i många datorsystem, så använder den fyra bytes. Hur höga värden kan man lagra i fyra bytes? För att räkna ut det, så säger man att på 8 bitar, så kan man lagra högst 2^8-1. Det blir 255. Något som vi kan se på bilden ovan. Fyra bytes är samma sak som 32 bitar. Alltså är det högsta värdet man kan lagra i 32 bitar 232-1 = 4294967295.

Det här är förstås en något förenklad förklaring, eftersom datorn också måste kunna hantera negativa tal och decimaltal. Jag kommer inte att gå in på det i den här boken, men den intresserade kan säkert hitta information om detta på Internet.

Vi kan väl kika på några bilder till. Du kan väl försöka räkna ut vad svaren blir innan du kikar på bilden under frågan.

Vad är 0110 1100?

0	1	1	0	1	1	0	0
	64	32	0	8	4	0	0
summa	108						

Vad blir 1001 1001?

1	0	0	1	1	0	0	1
128			16	8			1
summa	152						

Att göra om ett tal till en binärsträng

Vi kan väl utgå från att talet vi ska göra om, har ett värde mellan 0 och 255, På det viset vet vi att vi klarar oss med 8 bitar. Självklart, så ska vi skapa en funktion, som tar ett heltal som argument och som returnerar en sträng (string)

Vad har vi för data?

Vi har förstås ett heltal, som vi ska göra om till en sträng. Vi har en sträng, som vi bygger upp med ettor och nollor. Den strängen blir förstås utdata. Vi måste på något sätt ha tillgång till talserien, 1, 2, 4, 8, 16, 32, 64 och 128. Ett alternativ är att skapa en array, som håller talserien. Ett annat sätt är att ha en heltalsvariabel med värdet 128 och halvera den för varje varv i en iteration.

Vilka instruktioner är lämpliga?

I funktionen, som vi kan kalla intToBin får vi ett tal, som argument. I funktionen har vi två lokala variabler, en som vi kallar för bitposition, som är av heltalstyp och får värdet 128, en annan som vi kallar för retval, som är av typen string. I funktionen använder vi en while-loop, som körs så länge som bitposition delat på 2 är större än 0. Vi har också en lokal sträng i funktionen. I första varvet, så undersöker vi, om argumentet är större eller lika med bitposition. Skulle så vara fallet, så lägger vi en etta ”1” till retval. Vi minskar argumentets värde med bitposition. Skulle däremot argumentet vara mindre än bitposition, så lägger vi ”0” till retval. Längst ner i while-loopen halverar vi värdet på bitposition.

När iterationen avslutas, på grund av att värdet på bitposition inte är större än 0, så har vi binärsträngen i retval. Vi returnerar helt enkelt den.

int_to_binary.cpp

```
#include <cstdlib>

#include <iostream>

#include<string>
```

```
using namespace std;


string intToBin(int argument);
int main()
{
int tal = 65;
    cout<<intToBin(tal)<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
string intToBin(int argument)
{
string retval="";
int bitposition=128;
    while(bitposition>0)
    {
        if(argument>=bitposition)
        {
            argument-=bitposition;
            retval+="1";
        }
        else
            retval+="0";
    bitposition/=2;
    }
      return retval;
}
```

Det intressanta I programmet är förstås int_to_bin funktionen. Eftersom den finns utförligt beskriven under rubriken, ”vilka instruktioner är lämpliga?”, så har jag inte skrivit kommentarer i koden. Om du inte förstår den ändå, så rekommenderar jag, att du gör lite utskrifter i funktionen, så att du lättare kan följa, hur och varför binärsträngen byggs upp som den gör.

Programmet gav följande utskrift:

```
01000001

Tryck på en valfri tangent för att fortsätta...
```

Många datorer är utrustade med en kalkylator. Allt som oftast, så går dessa att växla mellan binärläge och decimaläge. Klistra gärna in resultatet från programkörningen i kalkylatorn, när den är i binärläge. Slå sedan om den till decimalläge och kontrollera resultatet. Observera, det kan vara så att kalkylatorn tar bort eventuella nollor, som ligger längst till vänster.

Jag kontrollerade resultatet från programmet i min kalkylator och kunde konstatera, att programmet hade fullgjort sin uppgift på ett tillfredsställande sätt.

Att göra om en binärsträng till ett tal

Jo, för att kunna lösa upp den binära texten, så måste vi på något sätt kunna översätta binärsträngar till heltal. Vi ska naturligtvis skapa en funktion, som tar en binärsträng som argument och returnerar ett heltal. Vi kan kalla funktionen binToInt.

Vad har vi för data att arbeta med?

Vi har förstås en binärsträng. Den ska bestå av åtta stycken ettor och eller nollor. Vi måste på något sätt ha tillgång till talserien, 1, 2, 4, 8, 16, 32, 64 och 128. Ett alternativ är att skapa en array, som håller talserien. Ett annat sätt är att ha en heltalsvariabel med värdet 128 och halvera den för varje varv i en iteration.

Vi måste ha en lokal variabel i funktionen, som håller värdet på binärsträngens tecken. Denna variabels värde ska returneras i slutet på funktionen och utgör därför utdata.

Dessutom, så måste vi ha en heltalsvariabel, som ska användas till att stega igenom tecken i binärsträngen. Eftersom vi har tänkt oss att gå från vänster till höger, så måste denna variabel initieras till 0.

Vilka instruktioner är lämpliga?

Om tecknet längst till vänster är en etta, så har den värdet 128. Nästa tecken har värdet 64 osv.

Man skulle alltså kunna tänka sig en while-loop liknande den vi hade, när vi gjorde om ett heltal till en binärsträng. I loopens första varv, så är bitpositionens värde 128. Om tecknet i binärsträngens första position (dvs. position 0) är en etta, så adderar vi 128 till den variabel, som ska hålla det sammanlagda bitvärdet. Om tecknet i första position är en nolla, så gör vi ingenting. Sist i while-loopen, så delar vi bitpositionens värde med två och får i första svängen 64. Vi räknar också upp variabeln, som anger positionen i binärsträngen med 1, vilket i första svängen innebär att den får värdet 1. När while-loopen terminerar (avslutas), så returneras värdet på variabeln, som håller bitarnas sammanlagda värden.

bin_to_int.cpp

```
#include <cstdlib>

#include <iostream>
```

```
#include <string>


using namespace std;
int binToInt(string bin);
int main()
{
string b ="01000001";
int tal = binToInt(b);
cout<<b<<"="<<tal<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}


int binToInt(string bin)
{
int bitposition = 128;//Bitpositionens värde
int bitvalue=0; //Ska hålla bitarnas sammanlagda värden.
int string_pos=0; //Används för att stega i strängen.


    while(bitposition>0)
    {
        if(bin[string_pos]=='1')
        /*På nästa rad räkas bitvärdet upp med 1,
          eftersom vi har stött på en etta i
          binärsträngen.*/
                bitvalue+=bitposition;
        bitposition/=2;//Halverar bitpositionen
        string_pos++;//Räknar upp strängstegaren med ett
    }
    return bitvalue;//Returnerar det sammanlagda värdet
}
```

En provkörning av programmet gav följande utskrift:

```
01000001=65

Tryck på en valfri tangent för att fortsätta...
```

En kontroll på min kalkylator visade, att programmet fungerat som tänkt.

Ett lite större program, som använder intToBin och binToInt

Längre fram i boken, närmare bestämt, när vi ska kika på filhantering, så ska vi göra ett program, som förvanskar en text och gör om den förvanskade texten till binärsträngar, som skrivs ner i en fil. Vi ska följa upp med ett annat program, som läser in binärsträngarna från en fil, gör om binärsträngarna till heltal, som görs om till den förvanskade texten, som görs om till läsbar text.

Innan dess, så ska vi göra om en text till binärsträngar, som sedan ska göras om till text igen.

Göra om Morgan till binärsträngar

Vad har vi för data att arbeta med?

Jo, vi har en textsträng med värdet Morgan. Vi ska också samla upp binärsträngarna i en stringarray. Eftersom det är 6 tecken i namnet Morgan, så måste vi ha plats med 6 tecken i stringarrayen. Vi kan använda en konstant, som vi kallar för SIZE och ställa den till värdet 6.

Hur ska man göra, om man inte på förhand vet, hur stort utrymme man behöver ha i en array? Det finns ett flertal olika lösningar på problemet. Man kan skapa arrayer med dynamiskt utrymme, men då måste man använda språkkonstruktionen pekare. Det finns också i standardbiblioteket dynamiska arrayer. Allt det där är saker, som längre fram kommer att gås igenom i boken.

Vilka instruktioner är lämpliga?

En idé är att iterera bokstäverna i strängen med hjälp av en for-loop. Varje tecken görs om till ett heltal och skickas till funktionen intToBin. Den sträng, som returneras, läggs i stringarrayen. Den kan vi iterera och skriva ut på skärmen.

bin_string_one.cpp

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;
string intToBin(int argument);
int main()
{
const int SIZE=6;
```

```
string meddelande="Morgan";
string binaries[SIZE];
    for(int i=0;i<SIZE;i++)
    {
        char c = meddelande[i];
        binaries[i] = intToBin((int)c);
    }

    for(int i=0;i<SIZE;i++)
            cout<<binaries[i]<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

string intToBin(int argument)
{
string retval="";
int bitposition=128;
    while(bitposition>0)
    {
        if(argument>=bitposition)
        {
            argument-=bitposition;
            retval+="1";
        }
        else
            retval+="0";
    bitposition/=2;
    }
      return retval;
}
```

Programmet gav följande tjusiga utskrift:

```
01001101

01101111

01110010

01100111

01100001

01101110

Tryck på en valfri tangent för att fortsätta...
```

Göra om binärsträngar till Morgan

Vad har vi för data att arbeta med?

Vi har en array med de binära strängarna i. Vi behöver ha en ny sträng, som ska användas till att sätta ihop de tecken, vi erhåller, genom att typkonvertera de tal vi får i retur från funktionen binToInt.

Vilka instruktioner är lämpliga?

I en for-loop itererar vi binärsträngarna. Varje sträng skickas till funktionen binToInt. Det tal vi får i retur från funktionen typkonverterar vi till en char, som konkateneras med den nya strängen. Eftersom det är en string, så kan vi konkatenera med +. Slutligen skriver vi ut den nya strängen på skärmen.

bin_string_two.cpp

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;
int binToInt(string bin);//Nytt, men du har koden sedan tidigare
string intToBin(int argument);
int main()
{
const int SIZE=6;
string meddelande="Morgan";
string binaries[SIZE];
    for(int i=0;i<SIZE;i++)
    {
```

```
        char c = meddelande[i];
        binaries[i] = intToBin((int)c);
    }

    for(int i=0;i<SIZE;i++)
            cout<<binaries[i]<<endl;
    /* Här nedan kommer nyheterna I programmet */
    cout<<"***********************"<<endl;
    string nystring="";
    for(int i=0;i<SIZE;i++)
    {
        int tal = binToInt(binaries[i]);
        char c = (char)tal;
        nystring+=c;
    }
    cout<<"nystring = "<<nystring<<endl;
/*Här är det slut på nyheterna */
    system("PAUSE");
    return EXIT_SUCCESS;
}

int binToInt(string bin)
{
int bitposition = 128;
int bitvalue=0;
int string_pos=0;

    while(bitposition>0)
    {
        if(bin[string_pos]=='1')
                bitvalue+=bitposition;
```

```
        bitposition/=2;
        string_pos++;
    }
    return bitvalue;
}
/* Lägg till funktionen här nedan */
string intToBin(int argument)
{
string retval="";
int bitposition=128;
    while(bitposition>0)
    {
        if(argument>=bitposition)
        {
            argument-=bitposition;
            retval+="1";
        }
        else
            retval+="0";
    bitposition/=2;
    }
      return retval;
}
```

Programmet gav följande imponerande utskrift:

```
01001101
01101111
01110010
01100111
01100001
01101110
```

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

nystring = Morgan

Tryck på en valfri tangent för att fortsätta...
```

Hänga gubbe

Ja, hänga gubbe är en lek, som du säkert har lekt någon gång. Själv så brukade min lärare på mellanstadiet göra det någon gång i emellanåt, när vi hade lite tid över. Då ritade han ett antal streck på tavlan, som markerade bokstäverna i ett ord han tänkte på. Det gällde för oss elever att gissa, vad ordet var, genom att fylla i bokstäver ovanför strecken. För varje gång, som vi gissade fel, så ritade magistern en del i en hägningsscen. Klarade vi inte av uppgiften, så hängdes gubben.

Jag tänkte att vi skulle göra ett program, som låter oss utföra precis samma sak. Man får ett antal stjärnor på skärmen, där varje stjärna representerar en bokstav, som ingår i ett ord, som datorn tänker på.

Vad har vi för data att arbeta med?

Vi har ett ord som ska gissas. Vi skulle kunna kalla det för computer_word. Vi ska ha en sträng, som är lika lång som computer_word och som består av stjärnor. Vi döper den till stars. Vi skapar en väl tilltagen array, som vi har för att lägga användaren prövade tecken i. Vi kan kalla den för user_tries. Vi behöver ha en heltalsvariabel, som håller reda på hur många tecken, som har lagts i user_tries. Vi kallar den för position. Vi behöver ha en heltalsvariabel, som ska hålla reda på antal försök användaren gjort. Vi kan döpa den till antal_inmatningar. Vidare så behöver vi har en bool, som vi kan ställa till false vid varje användarinmatning. Vi kan döpa den till finns. Tanken är att vi utgår från att tecknet inte har matats in av användaren sedan tidigare. Skulle vi hitta användarens inmatning, så ställer vi om finns, så att den får värdet true. Vi behöver ha en bool, som vi kan kalla för exists och som vi tilldelar värdet false, eftersom vi antar, att användarens tecken inte ingår i datorns ord.

Vilka instruktioner är lämliga?

En modell är att använda en while-loop, som itereras så länge, som användaren inte har gissat ordet eller tills denne har blivit hängd. I loopen uppmanas användaren att gissa, genom att skriva in ett tecken. Vi kollar i arrayen user_tries, om tecknet finns där sedan förut. Gör det inte det, så lägger vi in det där. Sedan deklarerar exists och vi ger den värdet false. Vi räknar upp position med ett. Sedan kollar vi om tecknet återfinns i datorns ord. Om så skulle vara fallet, så byter vi ut stjärnorna i stjärnsträngen på motsvarande position/positioner mot användarens tecken och ställer exists till true.

Skulle exists ha värdet false, så skriver vi ett meddelande till användaren, tecknet x ingår inte i datorns ord.

Om tecknet fanns i user_trials, så talar vi om det för användaren och uppmanar denne att mata in ett nytt tecken.

Vi kan i ett första försök låta användaren försöka skriva in 8 tecken. Lite senare ska vi komplettera med en liten mysig ritfunktion.

1. Skapa en sträng lika lång som datorns ord och som består av stjärnor

2. Startar en while-loop, som körs så länge, som stjärnsträngen inte är likadan som datorns ord eller om användaren har försökt 9 gånger.
3. Frågar användaren efter ett tecken
4. Ställer en bool (finns) till värde false.
5. I en for-loop kontrolleras användarens tidigare inmatningar (user_trials).
6. Om användarens tecken hittas, så ställs finns till true och for-loopen avslutas (break)
7. Om finns är false, så ökar vi värdet på antal_inmatningar med 1. Vi ställer exists till false, dvs. vi utgår från att tecknet inte finns i datorns ord. Vi går vi igenom datorns ord och matchar dess tecken mot användarens tecken.
8. Om användarens tecken finns i datorns ord, så byts stjärnsträngens stjärna ut i motsvarande position mot användarens tecken. Vidare så ändrar vi värdet på exists till true.
9. Om användarens tecken inte finns i strängen (exists är false), så skriver vi ut texten, tecknet x ingår inte i datorns ord. Längre fram, så ska vi rita lite på en hängande gubbe.
10. Vi lägger in tecknet i user_tries och räknar upp position med 1.
11. Om gubben inte är hängd eller om stjärnsträngen inte är lika med datorns ord, så hoppar vi till punkt 3.

hangman_one.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>
using namespace std;
/* clear används till att återställa inmatningsströmmen.*/
void clear();

int main()
{
dos_console();
string computer_word="dator";
string stars="";
stars.append(computer_word.length(),'*');
int position = 0;
char user_tries[28];
char c;
```

```
int antal_inmatningar = 0;
    while(computer_word != stars && antal_inmatningar<8)
    {
        cout<<stars<<endl;
        cin.get(c); //Istället för cin>>c;
        clear();
       bool finns=false;//Vi utgår från att tecknet inte har använts
            for(int i=0;i<position;i++)
            {
                if(user_tries[i]==c)
                {
                    finns=true;
                    break;
                }
            }
            if(!finns)
            {
                antal_inmatningar++;
                user_tries[position]=c;
                position++;
                bool exists=false;
                    for(int i=0;i<computer_word.length();i++)
                    {
                        if(computer_word[i]==c)
                        {
                            stars[i]=c;
                            exists=true;
                        }
                    }
          if(!exists)
            cout<<"Tecknet "<<c<<" ingår inte i datorns ord!"<<endl;
```

```
            }
            else
            {
                cout<<"Bokstaven "<<c<<" har redan använts."<<endl;
            }


    }
        cout<<computer_word<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

void clear()
{
     cin.clear();
     cin.ignore(100,'\n');
}
```

Ett utdrag från en provkörning kan du se nedan.

```
*****
p
Tecknet p ingår inte i datorns ord!
*****
p
Bokstaven p har redan använts.
*****
s
Tecknet s ingår inte i datorns ord!
*****
d
```

```
d****

a

da***

t

dat**

o

dato*

r

dator

Tryck på en valfri tangent för att fortsätta...
```

Vilken tur att jag visste vad datorn tänkte på. Annars, så hade jag väl aldrig klarat det. Nu ska vi rita lite. Någonting, som mina elever på IT-gymnasiet i Skövde brukar vara väldigt duktiga på, är att skapa bilder på skärmen med hjälp av de tecken, som finns att tillgå på tangentbordet. Två av mina elever i programmering B heter Alexander Gren och Philip Hassel. I årskurs 2, så skapade de en array av pekare till char, som om den skrivs ut på skärmen bildar en hängande gubbe. Jag har frågat Alexander och Philip, om jag får använda deras bildkonstruktion i den här boken. Det gick bra. Så ett stort tack till Alexander Gren och Philip Hassel.

VI ska börja med att skapa en funktion, som tar ett heltal som argument och skriver ut en bild på skärmen. Bilden varierar beroende på argumentets värde.

testa_gubbe.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;
void hangMan(int fas);

int main()
{
    for(int i=0;i<7;i++)
    {
        hangMan(i);
        cout<<"*************************"<<endl;
```

```
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}


void hangMan(int fas)
{
char *bild[7]={"|\n|\n|\n|\n|\n|\n",
     "________\n|\n|\n|\n|\n|\n|\n",
     "________\n|/\n|\n|\n|\n|\n|\n",
     "________\n|/     |\n|      |\n|\n|\n|\n|\n",
     "________\n|/     |\n|      |\n|      O\n|\n|\n|\n",
     "________\n|/     |\n|      |\n|      O\n|     -l-\n|\n|\n",
"________\n|/     |\n|      |\n|     O\n|     -l-\n|     / \\  \n|\n"};
cout<<bild[fas]<<endl;
}
```

En körning av programmet ger en serie utskrivna bilder. För varje bild blir hängningsscenen allt mer komplett.

Tanken är förstås att vi ska komplettera vårt program med funktionen hangman.

Istället för att skriva ut, ” Bokstaven x har redan använts”, så anropar vi funktionen hangman med antal_inmatningar som argument. Programmet lider dessutom av en liten felaktighet, som vi måste rätta till. Som det är nu, så räknas antalet_inmatningar upp med 1, om användaren har matat in ett tecken, som inte fanns i user_tries. Det innebär att user_tries räknas upp, oavsett om användaren har lyckats ange ett giltigt tecken eller inte. Om vi ska skriva ut en del av den hängande gubben, så är det väl rimligt att göra det, enbart om användaren har angett ett tecken, som inte ingår i datorns ord. Vi ska således bara räkna upp user_tries, om exists har värdet false. Jag anger detta med en kommentar i koden nedan.

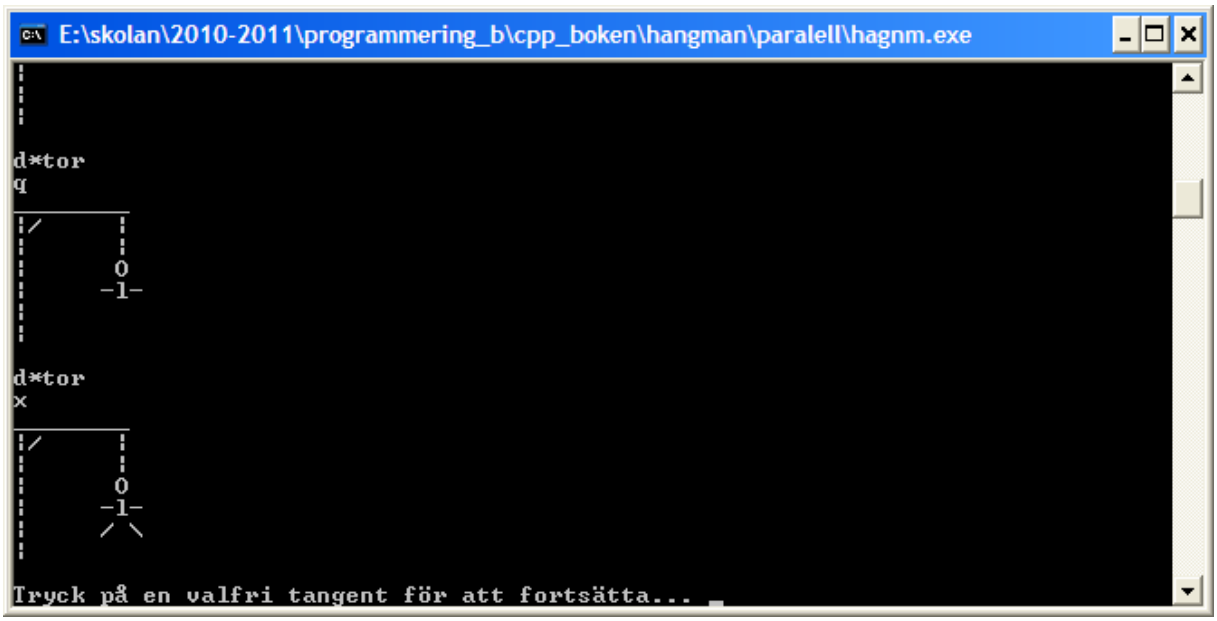
hangman_two.cpp

```
#include <cstdlib>
#include <iostream>
#include<string>
#include<iodos.h>
using namespace std;
void clear();
```

```
void hangMan(int fas);


int main()
{
dos_console();
string computer_word="dator";
string stars="";
stars.append(computer_word.length(),'*');
int position = 0;
char user_tries[28];
char c;
int antal_inmatningar = 0;
    while(computer_word != stars && antal_inmatningar<7)
    {
        cout<<stars<<endl;
        cin.get(c); //Istället för cin>>c;
        clear();
       bool finns=false;//Vi utgår från att tecknet inte har använts
            for(int i=0;i<position;i++)
            {
                if(user_tries[i]==c)
                {
                    finns=true;
                    break;
                }
            }
            if(!finns)
            {
                //antal_inmatningar++; Tas bort härifrån
                user_tries[position]=c;
                position++;
```

```
                bool exists=false;
                    for(int i=0;i<computer_word.length();i++)
                    {
                        if(computer_word[i]==c)
                        {
                            stars[i]=c;
                            exists=true;
                        }
                    }
                    if(!exists)
                    {
                        hangMan(antal_inmatningar);
                        antal_inmatningar++;//Placeras här
                    }
            }
            else
            {
                cout<<"Bokstaven "<<c<<" har redan använts."<<endl;
            }
       }

    system("PAUSE");
    return EXIT_SUCCESS;
}

void clear()
{
     cin.clear();
     cin.ignore(100,'\n');
}
```

```
void hangMan(int fas)
{
char *bild[7]={"|\n|\n|\n|\n|\n|\n",
"_______\n|\n|\n|\n|\n|\n|\n",
"_______\n|/\n|\n|\n|\n|\n|\n",
"_______\n|/     |\n|      |\n|\n|\n|\n|\n",
"_______\n|/     |\n|      |\n|      O\n|\n|\n|\n",
"_______\n|/     |\n|      |\n|      O\n|     -l-\n|\n|\n",
"_______\n|/     |\n|      |\n|      O\n|     -l-\n|     / \\ \n|\n"};

        cout<<bild[fas]<<endl;
}
```



Oj då! Så kan det tydligen också gå.

Rekursion

Tidigare i boken, så listade jag upp de olika sätt, som kod kan exekveras på. Dessa var sekvens, selektion, iteration och rekursion. Man kan också säga att dessa är algoritmens byggstenar. Jag gick sedan igenom alla byggstenarna, utom rekursion. För att kunna förstå rekursion, så måste man också förstå sig på funktioner. Det är därför som rekursionsavsnittet dyker upp först här.

Rekursion är kanske den svåraste byggstenen i algoritmen och för många programmerare, så är det den man behärskar sist. Men, det är mödan värt, eftersom den ofta leder till eleganta och effektiva lösningar. Samtidigt ska påpekas, att man för det allra mesta kan lösa samma problem, utan att

tillgripa rekursion, vilket också leder till en längre och senare inlärning. Med detta vill jag ha sagt, bli inte förskräckt, om du tycker att det känns svårt med rekursion. Du kommer med tiden att lära dig och innan dess, så kan du faktiskt klara sig riktigt bra utan.

För många programmerare, så brukar den första kontakten med rekursion vara helt oplanerad och ett misstag. Man råkar på något sätt direkt eller indirekt göra ett rekursivt anrop, som låser programmet, eftersom man inte har erbjudit en väg ut ur rekursionen.

evig_rekursion.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

void funktion1();
void funktion2();

int main()
{
    funktion1();
    system("PAUSE");
    return EXIT_SUCCESS;
}

void funktion1()
{
    cout<<"Funktion 1"<<endl;
    funktion2();
}

void funktion2()
{
    cout<<"Funktion 2"<<endl;
    funktion1();
```

```
}
```

Om du studerar programmet ovan, så kan du se, att det sker ett anrop till funktion1 från mainfunktionen. I funktion1 sker ett anrop till funktion2. I denna sker i sin tur ett anrop till funktion1. När funktioner anropar varandra på ett rekursivt sätt, så kallas det för en indirekt rekursion. I det här fallet, så finns det inte i koden någon utväg ur rekursionen, utan programmet har kört fast. Vid en provkörning av programmet, så erhölls en aldrig avstannande utskrift av, Funktion 1, Funktion 2, Funktion 1 osv.

Nästa misstag, fel, som visas är betydligt allvarligare. I det här fallet, så handlar det om en direkt rekursion.

allvarligt_fel_rekursion.cpp

```
#include <iostream>


using namespace std;


int funktion();


int main()
{
    cout<<funktion();
    system("PAUSE");
    return EXIT_SUCCESS;
}


int funktion()
{
    return 5+funktion();
}
```

Nu ska funktion returnera ett heltal. I mainfunktionen görs ett försök att skriva ut returvärdet från funktionen. I funktion sker sedan ett rekursivt anrop till funktion. Return 5+funktion();

Inte i det här fallet heller finns det i koden någon möjlighet för rekursionen att upphöra. Nu, kan det bli tal om en allvarligare programkrasch. När jag körde programmet i DevC++, så stängde programmet ner omedelbart. I andra miljöer, så kanske det inte slutar med en lika lycklig utgång.

Att lära av dessa misslyckade rekursioner är, att vi måste skriva kod, som erbjuder en väg ut ur rekursionen.

Funktionen skrivTecken

Vi ska börja med ett enkelt exempel. En funktion, som tar en char och ett heltal, som argument. Funktionen ska heta skrivTecken och meningen är att funktionen ska skriva ut första argumentet så många gånger, som anges av andra argumentet. Vi har tidigare gjort en sådan funktion, men nu ska vi använda den rekursiva idén.

skriv_tecken1.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

void skrivTecken(char c,int antal);

int main()
{
    skrivTecken('*',5);
    cout<<endl;
    system("pause");
    return EXIT_SUCCESS;
}

void skrivTecken(char c,int antal)
{
    if(antal>1)//Först det enkla fallet
        skrivTecken(c,antal-1);
        cout<<c;
}
```

Om vi studerar koden I funktionen skrivTecken, så upptäcker vi att det finns kod I den, som borgar för att rekursionen upphör. Det är nämligen så, att ett rekursivt anrop till funktionen sker endast om antal är större än 1. Eftersom antal minskas med 1 vid varje anrop, så kommer de rekursiva anropen att upphöra.

Programmet gav följande utskrift:

```
*****
```

```
Tryck på en valfri tangent för att fortsätta...
```

Nu ska vi modifiera koden en aning. Syftet är att utforska lite närmare, vad det är som händer i funktionen. Skriv om funktionen skrivTecken, så att den ser ut så här:

```
void skrivTecken(char c,int antal)
{
    if(antal>1)//Först det enkla fallet
        skrivTecken(c,antal-1);
        cout<<"antal= "<<antal<<" "c;
}
```

Tanken bakom det hela är att vi ska se värdet på antal. Programmet ger nu följande utskrift:

```
antal= 1 *antal= 2 *antal= 3 *antal= 4 *antal= 5 *
Tryck på en valfri tangent för att fortsätta...
```

Förvånad!? Första anropet utförs uppenbarligen sist. Vad som händer vid ett rekursivt anrop är att funktionen skrivTecken i det här fallet finns i ett antal upplagor, kopior. Varje anrop får sin kopia. Vid första anropet har antal värdet 5, men efterson 5 är större än 1, så sker ett nytt anrop, där antal har fått värdet 4. Så där fortsätter det tills antal får värdet 1, vilket sker i det sista anropet. Då exekveras resten av funktionen i det sista anropet, då antal är lika med 1. Sedan backar exekveringen i samma ordning, som anropen skedde. Det innebär att nästa funktion, som exekveras, är den där värdet på antal är lika med 2. Det beror naturligtvis i det här fallet också på att det rekursiva anropet sker innan utskriften.

Frågan är då, hur man ska göra i funktionen, om man vill att den ska ta ansvar för utskriften av radbrytet. Om du tittar i källkoden, så kan du se att radbrytet ligger i mainfunktionen.

```
skrivTecken('*',5);
cout<<endl;
```

Jo, ett sätt skulle kunna vara att först skriva ut tecknet i funktionen, dvs. innan man prövar värdet antal. Om antal då är större än 0, så skrivs tecknet ut och ett rekursivt anrop sker, annars så skrivs ett radbryt ut.

Se den modifierade funktionen nedan:

```
void skrivTecken(char c,int antal)
{

    if(antal>0)
    {
```

```
        cout<<c;
        skrivTecken(c,antal-1);
    }
    else
        cout<<endl;
}
```

Funktionen upphojt

Vi ska nu titta på ett annat exempel, där ett rekursivt anrop erbjuder en elegant lösning. Om man skulle vilja beräkna $2^{2,}$ så finns det förstås en funktion i mattebiblioteket, som heter pow. Funktionen tar två argument, basen och exponenten. Men jag tänkte att vi skulle skriva en egen funktion, som gör jobbet. 2^2 är samma sak som 2*2. 2^3 är samma sak som 2*2*2. Vi kan ju börja med att lösa uppgiften, utan att använda rekursion.

upphojt.cpp

```
#include <cstdlib>
#include <iostream>

using namespace std;

double upphojt(double base,double exponent);

int main()
{
    cout<<upphojt(4,4)<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

double upphojt(double base,double exponent)
{
    double retval=1;
    int start = 0;
```

```
            while(start<exponent)
            {
                retval*=base;
                exponent--;
            }
        return retval;
}
```

Jag tror att lösningen är både rättfram och enkel. Kanske hade du kunnat lösa det på ett liknande sätt. Vi ska väl kika på en utskrift från en testkörning innan vi fortsätter.

```
256
Tryck på en valfri tangent för att fortsätta...
```

En snabb kontroll med kalkylatorn visar att funktionen fungerar som den ska.

Nu ska vi göra samma sak, men nu ska vi använda rekursion. I funktionen upphojt, så börjar vi med det enkla fallet, nämligen att exponent har värdet 1. Skulle så vara fallet, så returneras base gånger 1.

Om exponent skulle vara större än ett, så returneras base * ett rekursivt anrop med exponent minskat med 1. Slutresultatet blir detsamma som i den ”vanliga” lösningen ovan.

```
#include <cstdlib>
#include <iostream>



using namespace std;

double upphojt(double base,double exponent);

int main()
{
    double t = upphojt(16,3);
    cout<<t<<endl;
    system("pause");

    return EXIT_SUCCESS;
```

```
}


double upphojt(double base,double exponent)
{
    if(exponent==1)
            return base*1;
    return base * upphojt(base,exponent-1);
}
```

En liten test av programmet gav:

```
4096

Tryck på en valfri tangent för att fortsätta...
```

En kontroll med kalkylatorn visade att funktionen fungerade som det ska.

Avslutning rekursion

Jag förstår, om du tycker att det här avsnittet om rekursion känns jobbigt och komplicerat. Det är sant, rekursion är komplicerat och är väl kanske inte det första man arbetar med, när man lär sig programmering. Vi kunde ovan också se, att man kan lösa uppgiften med en vanlig algoritm.

Mitt tips till dig är, att träna på den rekursiva programmeringsmodellen någon gång i emellanåt. Du behöver inte känna dig misslyckad, om du inte förstår och behärskar den med en gång. Så småningom, så kommer du att använda den, men skynda långsamt, om du tycker att det verkar jobbigt.

Att dela upp program i fler filer

Jag ska visa vad jag menar med med det som sägs i rubriken med ett exempel. Vi ska helt enkelt bygga ett litet funktionsbibliotek, vars funktioner hanterar en array av doubles.

När man delar upp program i flera olika filer, så brukar man lägga funktionsprototyperna i en fil för sig, en headerfil, och själva funktionerna i en källkodsfil (många gånger med efternamnet cpp).

Vi kan väl börja med en funktion, som skriver ut innehållet i en array. Vi börjar med headerfilen.

array_funktioner.h

```
#ifndef _ARRAY_FUNKTIONER_H
#define _ARRAY_FUNKTIONER_H
#include<iostream>
```

```
using namespace std;


/*skrivArray skriver ut innehållet i en array*/
void skrivArray(double arr[],int storlek);


#endif
```

Du funderar säkert över de två första raderna och den sista., #ifndef osv. Låt mig börja med att lugna dig med att dessa rader inte är skrivna i c++. De är skrivna i ett slags makrospråk och utgör en instruktion till kompilatorn. I det här fallet, så betyder de, att om filen redan är kompilerad, så bry dig inte om att kompilera in den i projektet en gång till. I mindre projekt, så är risken inte särskilt stor att man råkar länka in samma fil mer än en gång, men det är lika bra att du lär dig rätt från början, eller hur? Ifndef är förstås kort för engelskan if not defined (om inte definierad) . Define betyder förstås definiera och slutligen, så betyder endif, slut på om. I det här fallet, och i alla de fall framöver i boken, där jag arbetar med fler filer i samma program, så kommer jag att använda filens namn, i det här fallet _ARRAY_FUNKTIONER_H, som villkor i ifsatsen. Det är inte alls nödvändigt att göra på det viset, jag hade kunna välja att skriva ISGLASS istället, men det är en god vana att skriva på ett beskrivande sätt.

I övrigt, så känner du igen skrivsättet för en funktionsprototyp. Jag har dessutom skrivit en kommentar ovanför funktionen skrivArray. Dessutom så har funktionen också fått ett beskrivande namn. Varför ska man lägga någon energi på det då? Jo, om någon annan skulle använda mitt bibliotek, så är det inte säkert att jag skickar med källkoden, utan kanske en kompilerad objekts eller biblioteksfil samt headerfilen. Då utgör headerfilen den enda källa till information, om hur mina funktioner ska användas. Det är samma sak för din del, om du vill veta mer om ett bibliotek, kanske ett som du hittar i din kompilators installationsmappar. Du är hänvisad till headerfilen för att se hur biblioteket ska användas.

Jo. precis som förut, så måste vi skriva själva källkoden.

array_funktioner.cpp

```
#include "array_funktioner.h"


void skrivArray(double arr[],int storlek)
{
    for(int i=0;i<storlek;i++)
        cout<<"arr["<<i<<"]="<<arr[i]<<endl;
}
```

Det enda som kanske förvånar här är att <> vid inkluderingen har försvunnit. Hakarna innebär att kompilatorn letar i kompilatorinstallationens inkluderingsmappar. Gör man en inkludering från den projektmapp man arbetar i, så räcker det med filens namn.

Ska vi testa att använda funktionsbiblioteket i ett program då? Man inkluderar även här utan hakar, eftersom det är en lokal fil som inkluderas.

funktions_testare.cpp

```
#include <cstdlib>
#include <iostream>
#include "array_funktioner.h"

using namespace std;

int main()
{
    double provresultat[]=
    {34.2,15.7,17.2,29.2,39.0,
     45.4,18.6,32.3,35.5,11.2,
     43.5,27.2,28.5,31.5,33.0};
    int storlek = sizeof(provresultat)/sizeof(double);

    skrivArray(provresultat,storlek);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Jag tror inte att jag behöver förklara några detaljer I det här programmet, eftersom du redan kan det här med arrayer. Men jag tror nog att programmet klargör de funktionsanrop, du gjort i tidigare program. Enda skillnaden mellan de programmen och det här är, att det är du som har skapat funktionen som anropas. Till sist testar vi om det fungerar som tänk.

```
arr[0]=34.2
arr[1]=15.7
arr[2]=17.2
arr[3]=29.2
```

```
arr[4]=39
arr[5]=45.4
arr[6]=18.6
arr[7]=32.3
arr[8]=35.5
arr[9]=11.2
arr[10]=43.5
arr[11]=27.2
arr[12]=28.5
arr[13]=31.5
arr[14]=33
Tryck på en valfri tangent för att fortsätta...
```

Prydligt eller hur?

Vi bygger vidare. En funktion, som returnerar medelvärdet av provresultaten

En funktion, som skriver ut en array på ett smakfullt, var väl ingen utmaning eller hur? Vi gjorde den ju tidigare i boken. Enda skillnaden var att vi skrev funktionen och programmet i samma fil. Samma gäller förstås konststycket att räkna ut en arrays medelvärde, men lite repetition skadar aldrig.

Vad har vi för data att arbeta med?

I funktionen, så har vi en array med ett antal värden. Vi har också tillgång till antal element i arrayen, en nödvändighet, om vi ska kunna iterera arrayens samtliga element. Vi ska ha en variabel, som håller summan. Som utdata kan vi använda en variabel, men vi kan också returnera ett aritmetiskt uttryck, return summa/storlek.

Vilka instruktioner är lämpliga?

Vi måste på något sätt räkna ut summan av de i arrayen ingåendeelementen. En for-loop skulle säkert fungera. Sedan delar vi den framräknade summan med antalet element.

Vi börjar med att lägga till funktionsprototypen i **array_funktioner.h**

```
/*getAverage returnerar argumentarrayens medelvärde */
double getAverage(double arr[],int storlek);
```

Så hoppar vi över till källkodsfilen **array_funktioner.cpp**

```
double getAverage(double arr[],int storlek)
{
double summa=0;
    for(int i=0;i<storlek;i++)
        summa+=arr[i];
    return summa/storlek;
}
```

Då återstår väl egentligen bara att testa funktionen. Jag har lagt till en inkludering av iodos.h och ett anrop till funktionen dos_console, och självklart ett anrop till vår nya funktion.

funktions_testare.cpp

```
#include "array_funktioner.h"

using namespace std;

int main()
{
    dos_console(); //Obs nytt
    double provresultat[]=
    {34.2,15.7,17.2,29.2,39.0,
     45.4,18.6,32.3,35.5,11.2,
     43.5,27.2,28.5,31.5,33.0};
    int storlek = sizeof(provresultat)/sizeof(double);

    skrivArray(provresultat,storlek);
    double medel = getAverage(provresultat,storlek);
    cout<<"Medelresultatet på provet är "<<medel<<" poäng."<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En provkörning gav följande resultat:

```
Medelresultatet på provet är 29.4667 poäng.
```

```
Tryck på en valfri tangent för att fortsätta...
```

Jag utelämnade utskriften av arrayens innehåll. Nu ska man egentligen plocka fram sin miniräknare och kontrollräkna resultatet. Ett annat sätt att kontrollera resultatet är förstås att förenkla arrayen.

enkelt_test.cpp

```
double provresultat[]= {12.0,6.0,3.0};
```

I övrigt som i funktionstestare.cpp

Summan av arrayen är 21 och 21/3 = 7. Som tur är så gav en provkörning av programmet följande utskrift:

```
arr[0]=12

arr[1]=6

arr[2]=3

Medelresultatet på provet är 7 poäng.

Tryck på en valfri tangent för att fortsätta...
```

Med detta vill jag visa, att det är mycket viktigt att man testar och kontrollerar sina funktioner, så att de fungerar som de ska.

Du kanske tyckte att utskriften, ”Medelresultatet på provet är 29.4667 poäng”, har för många decimaltal. Vi ska längre fram i boken kika på olika modeller för styrning av utskrifter, men jag kommer redan nu att visa, hur man skulle kunna göra, för att fixa till den här utskriften. Det finns någonting, som kallas för utskriftsmanipulatorer. Dessa anges i utskriftsströmmen, t.ex. så finns en manipulator som heter fixed.

```
double d = 45;
    cout<<fixed<<d<<endl;
```

Ger utskriften 45.000000, dvs. med decimaltalsdel. Då kan man med funktionen setprecision ange, hur många tal man vill ha i decimaldelen. Funktionsanropet måste ske på följande sätt:

cout<< fixed<<setprecision(2)<<d<<endl; . Uttrycket skulle ge utskriften, 45.00

För att få det hela att fungera, så måste man också inkludera filen **iomanip.p.**

Vi fixar till programmet, som testar våra funktioner **.**

funktions_testare.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<iomanip> //OBS nytt
```

```
#include "array_funktioner.h"


using namespace std;


int main()
{
    dos_console();


    double provresultat[]=
    {34.2,15.7,17.2,29.2,39.0,
     45.4,18.6,32.3,35.5,11.2,
     43.5,27.2,28.5,31.5,33.0};


    int storlek = sizeof(provresultat)/sizeof(double);


    skrivArray(provresultat,storlek);
    double medel = getAverage(provresultat,storlek);
    cout<< setiosflags(ios::fixed)<<setprecision(2)<<
   "Medelresultatet på provet är "<<
    medel<<" poäng."<<endl; //OBS Nytt
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En provkörning gav följande resultat:

```
Medelresultatet på provet är 29.47 poäng.

Tryck på en valfri tangent för att fortsätta...
```

Jag utelämnade utskriften av arrayens innehåll. Men visst blev det väl prydligt. *Vi ska som sagt titta på manipulering av utskrifter längre fram i boken, då kommer vi se att det finns fler olika alternativ att åstadkomma samma resultat.*

Vi bygger vidare. En funktion, som returnerar det högsta av provresultaten

Vi har faktiskt även kikat på detta lite tidigare i boken. Men nu ska vi ju skapa en funktion, som löser uppgiften, så vi passar på att repetera lite.

Vad har vi för data?

Jo, som argument till funktionen, så har vi provresultaten i form av en array. Vi har också en variabel, som anger antalet element i arrayen. Vi bör ha en variabel, som håller det högsta resultatet i arrayen. Vi kan kalla den för best. Värdet på denna variabel ska returneras av funktionen. Den kan kategoriseras som funktionens utdata.

Vilka instruktioner är lämpliga?

Jo enkelt uttryckt, vi utgår från att arrayens första element också är dess högsta. Sedan går vi igenom resten av arrayens element och jämför med värdet på variabeln, som håller det hittills högsta provresultatet. Skulle något av arrayens element ha ett högre värde, så sätter vi värdet på variabel, som ska hålla det högsta värdet, till elementets värde. Vi kan punkta algoritmen enligt nedan:

Anta att första elementet i arrayen är högst och tilldela variabeln best detta värde

- Iterera arrayen med start från 1
- Jämföra elementet med best.
- Om element har ett högre värde än best, så får best detta värde
- Returnera värdet på best

Komplettera **array_funktioner.h** med följande:

```
/*getBest returnerar argumentarrayens högsta värde */
double getBest(double arr[],int storlek);
```

Sedan hoppar vi till **array_funktioner.cpp** och skriver källkoden till funktionen.

```
double getBest(double arr[],int storlek)
{
double best=arr[0];
    for(int i=1;i<storlek;i++)
    {
        if(arr[i]>best)
            best = arr[i];
    }
   return best;
```

```
}
```

Och självklart, så ska vi utöka vårt testprogram med en test av vår nya funktion.

funktions_testare.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<iomanip>
#include "array_funktioner.h"

using namespace std;

int main()
{
    dos_console();

    double provresultat[]=
    {34.2,15.7,17.2,29.2,39.0,
     45.4,18.6,32.3,35.5,11.2,
     43.5,27.2,28.5,31.5,33.0};

    int storlek = sizeof(provresultat)/sizeof(double);

    skrivArray(provresultat,storlek);
    double medel = getAverage(provresultat,storlek);
    cout<<setiosflags(ios::fixed)<<setprecision(2)
    <<"Medelresultatet på provet är "<< medel<<" poäng."<<endl;
    cout<<"Högsta provresultatet är "<<
        getBest(provresultat,storlek)<<" poäng"<<endl;//OBS Nytt
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift:

```
Medelresultatet på provet är 29.47 poäng.

Högsta provresultatet är 45.40 poäng

Tryck på en valfri tangent för att fortsätta...
```

Jo funktionen fungerar som den ska. Jag har som vanligt utelämnat utskriften av arrayen.

Om vi tänker oss att läraren i fråga är lite elak, så kan han ha satt upp en regel, som säger att man måste ha minst en poäng, som överstiger medelvärde med tre, för att få godkänt på provet. Jag tror inte skolverket skulle uppskatta konstruktionen, men hur ska vi lösa uppgiften i vårt program?

En funktion som skriver ut resultatet och om resultatet är godkänt eller underkänt

Vi har som tur är redan en funktion, som returnerar medelvärdet. Den ska vi naturligtvis använda för att hämta hem medelvärdet. Funktionen, som vi kan kalla för testaProvresultat, ska inte returnera någonting. Den ska ju endast skriva ut alla provresultat och avgöra om det ska skrivas ut godkänt eller underkänt. Den ska ha tre argument, arrayen med provresultatet, ett heltal, som anger hur många provresultat det finns, samt en double som anger gränsvärdet för godkänt. Vi kan lägga till funktionsprototypen i headerfilen.

array_funktioner.h (fortsättning)

```
/*testaProvresultat skriver ut samtliga elementen i argumentarrayen.
  Efter en prövning av om elementets värde är större eller lika med
  gränsvärdet -limit, så skrivs godkänt eller underkänt ut*/
void testaProvresultat(double arr[],int storlek,double limit);
```

Då återstår att lösa uppgiften och skriva källkod.

array_funktioner.cpp (fortsättning)

```
void testaProvresultat(double arr[],int storlek,double limit)
{
        for(int i=0;i<storlek;i++)
        {
            cout<<"Provresultat: "<<arr[i];
            if(arr[i]>=limit)
                    cout<<" Godkänt.";
            else
                    cout<<" Underkänt.";
```

```
            cout<<endl;
        }
}
```

Ja, då är det väl bara att testa funktionen I vårt testprogram.

funktions_testare.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<iomanip>
#include "array_funktioner.h"
using namespace std;
int main()
{
    dos_console();
    double provresultat[]=
    {34.2,15.7,17.2,29.2,39.0,
     45.4,18.6,32.3,35.5,11.2,
     43.5,27.2,28.5,31.5,33.0};

    int storlek = sizeof(provresultat)/sizeof(double);

    skrivArray(provresultat,storlek);
    double medel = getAverage(provresultat,storlek);
    cout<<setiosflags(ios::fixed)<<setprecision(2)
    <<"Medelresultatet på provet är "<< medel<<" poäng."<<endl;

    cout<<"Högsta provresultatet är "<<
        getBest(provresultat,storlek)<<" poäng"<<endl;
    double limit = medel-3; //OBS nytt
    testaProvresultat(provresultat,storlek,limit); //OBS nytt
        system("PAUSE");
```

```
    return EXIT_SUCCESS;
}
```

En testkörning gav följande resultat. Den första utskriften av arrayens innehåll har utelämnats.

```
Medelresultatet på provet är 29.47 poäng.

Högsta provresultatet är 45.40 poäng

Provresultat: 34.20 Godkänt.

Provresultat: 15.70 Underkänt.

Provresultat: 17.20 Underkänt.

Provresultat: 29.20 Godkänt.

Provresultat: 39.00 Godkänt.

Provresultat: 45.40 Godkänt.

Provresultat: 18.60 Underkänt.

Provresultat: 32.30 Godkänt.

Provresultat: 35.50 Godkänt.

Provresultat: 11.20 Underkänt.

Provresultat: 43.50 Godkänt.

Provresultat: 27.20 Godkänt.

Provresultat: 28.50 Godkänt.

Provresultat: 31.50 Godkänt.

Provresultat: 33.00 Godkänt.

Tryck på en valfri tangent för att fortsätta...
```

Hårda villkor, javisst, men en enklare kontroll visar att vår strategi fungerade. *Vi återkommer till provresultat längre fram i boken, då vi också ska ha med elevernas namn.*

Pekare – primärminnesadresser och referenser

Du har vid åtskilliga tillfällen sett hur vi har använt variabler av olika slag.

```
int tal=5;
```

Här deklareras en variabel av heltalstyp, som tilldelas värdet 5. Det innebär också att ”tal” fungerar som en platshållare. Egentligen är det så att ”namn” är ett alias för en primärminnesadress. Vilken adress, spelar normalt sett inte något som helst roll, vare sig för programmet eller för

programmeraren. Variabeln tal kommer automatiskt att tilldelas en plats i primärminnet, där det för tillfället finns plats. Det innebär att adressen till tal kan variera från dator till dator och från körning till körning. Men man kan faktiskt ta reda på vilken primärminnesadress som används.

pekare_ett.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;

int main()
{
int tal=5;
    cout<<"adressen till tal: "<<&tal<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

För att få reda på adressen till en variabel, så använder man adressoperatorn. Programmet ovan gav följande utskrift:

```
adressen till tal: 0x22ff74
Tryck på en valfri tangent för att fortsätta...
```

Adressen uttrycks med ett hexadecimalt tal. Det är ett talsystem med 16, som talbas. Jämför med det tvåvärda, binära talsystemet.

Med en utskriftsmanipulator, så skulle man kunna göra om den hexadecimala utskriften till utskrift av ett heltal.

(Vi kommer att gå igenom formatering av utskrifter längre fram i boken)

```
#include <cstdlib>
#include <iostream>
#include<iomanip>

using namespace std;

int main()
```

```
{
int tal=5;
    cout<<"adressen till tal: "<<dec<<(int)&tal<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Vilket ger utskriften:

```
adressen till tal: 2293620

Tryck på en valfri tangent för att fortsätta...
```

Då ser vi tydligt att tal låg på adress 229936620 eller 0x22ff74, när programmet kördes.

I C++ finns det en särskild variabeltyp, som arbetar med primärminnesadresser. Dessa variabler kallas för pekare. Man kan deklarera en pekare på följande sätt:

```
int *pek;
```

Det här betyder att vi har en pekare till ett heltal. Pekaren ovan pekar inte på någonting ännu.

```
/*Man kan också deklarera en pekare, som inte pekar på något bestämt
för tillfället, en s.k. nollpekare. Dessa kan vara väldigt
användbara i många program. Om man inte från tillfälle till
tillfälle kan veta, om pekaren pekar på något, så kan man testa med
if(pekare != 0) gör något.*/

int *pek=0; //En så kallad nollpekare.
```

Man kan deklarera flera variabler på en rad, enligt följande:

```
int tal,number;
```

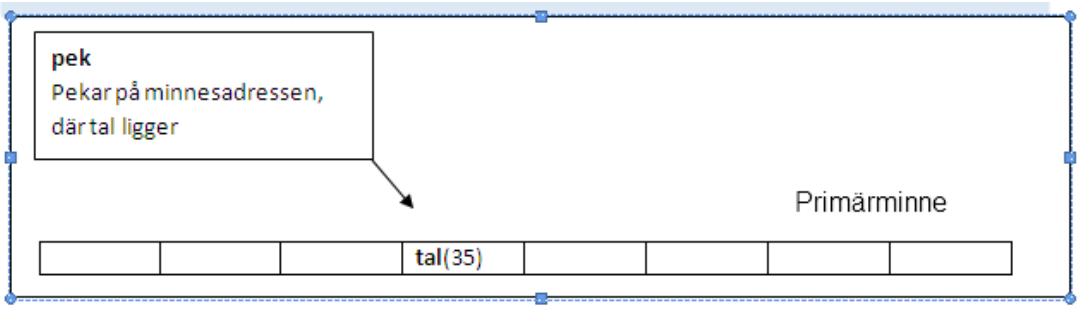
Raden ovanför innebär att vi får två heltalsvariabler, som heter tal och number. När det gäller pekare, så får man lov att se upp lite grann.

```
int *pek,number; //Här är det lätt att göra misstag.
```

Raden ovanför betyder inte, vilken man skulle kunna tro, att vi har fått två pekarvariabler, som heter pek och number. Vi har fått en pekarvariabel, som heter pek och en heltalsvariabel, som heter number.

```
int tal=35; //tal tilldelas värdet 35

pek = &tal;//pek pekar på den adress, som tal ligger på,
```

Om man tänker sig minnet, som en lång rad av fack, som man kan stoppa saker och ting i, så kan man göra en bild, som beskriver förhållandet enligt nedan.



Vi ska naturligtvis göra ett litet program, som demonstrerar en pekare.

pekare_one.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
int tal = 5; //vanlig variabel
int* pek; //En pekarvariabel

    pek = &tal; //pek pekar på tals minnesadress
    cout<<"pek pekar på minnesadress: "<<pek<<endl;
    cout<<"På den minnesadress, som pek pekar på"<<
          " finns värdet "<<*pek<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift. Observera, att det inte är säkert, att du får exakt samma utskrift, eftersom minnesadressen kan variera.

```
pek pekar på minnesadress: 0x22ff74
```

```
På den minnesadress, som pek pekar på finns värdet 5

Tryck på en valfri tangent för att fortsätta...
```

Eftersom variabeln tal är ett alias för en primärminnesadress och pek pekar på denna, så skulle man kunna säga, att pek och tal är alias för vartannat. Vi ska se på två korta program, som visar hur det kan komma till uttryck.

pekare_two.cpp

```
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
int tal = 5; //vanlig variabel
int* pek; //En pekarvariabel

    pek=&tal;//Pek pekar på tals adress
    cout<<"tal="<<tal<<endl;
    *pek=5;//Vi ökar det värde, som pek pekar på
    cout<<"tal="<<tal<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande överraskande utskrift:

```
tal=5

tal=25

Tryck på en valfri tangent för att fortsätta...
```

Det som överraskar är förstås att värdet på tal har ändrats, utan att vi har gjort någonting alls med tal. Istället använde vi pekaren, för att förändra det som pek pekar på, vilket är samma ställe, som tal har sitt värde på. Det är således inte särskilt överraskande, men håll med om att det ser skumt ut.

Frågan är då, om vi kan göra tvärtom? Ändra på det värde, som ligger på den adress, som pekaren pekar på, via en variabel?

pekare_three.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

int main()
{
dos_console();
int tal = 5; //vanlig variabel
int* pek; //En pekarvariabel

    pek=&tal;//Pek pekar på tals adress
    cout<<"pek pekar på värdet:"<<*pek<<endl;
    tal/=5; //Dividerar tal med 5
    cout<<"pek pekar på värdet:"<<*pek<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Jorå, utskriften från programmet visar, att det fungerade också från andra hållet.

```
pek pekar på värdet:5
pek pekar på värdet:1
Tryck på en valfri tangent för att fortsätta...
```

Pekare till konstanter och konstanta pekare

Med nyckelordet const kan man skapa konstanter i C++. Det visste du förstås sedan tidigare. Enkelt uttryckt, så är en konstant en variabel, som man inte kan ändra på. Det är därför, som man måste tilldela den ett värde redan från början.

```
const int MAX=340;
MAX = 122; //Otillåtet
```

I samband med pekare, så kan man använda const på två sätt.

Man kan använda låta pekaren peka på ett konstant objekt. Då kan man inte använda pekaren till att ändra på det utpekade värdet.

```
double pris=14.50;
const double *pekare = pris;
```

Raderna ovan säger att man inte kan använda pekaren till att ändra det utpekade värdet. Följande kodrader är således felaktiga.

```
*pekare = 18.50; //fel
cin>>*pekare; //fel
```

Men däremot, så kan man ändra värdet via variabeln pris, eftersom pris inte är en konstant.

```
pris = 27.0; //Helt okey
```

En pekare till en konstant kan peka på ett konstant värde och ett vanligt ändringsbart värde. Men en vanlig pekare kan inte peka på ett konstant värde.

Det andra sättet att använda const på är att ha en konstant pekare. Det innebär att själva pekaren inte får ändras.

```
double pris=13.50;
double nytt_pris=0.50;
double *const pekare = &pris;
pekare = &pris; //Otillåtet
pris=11.50;//Ok
```

Konstanta pekare i funktionens parameterlista

Det rekommenderas, att om man kan använda en konstant pekare, som argument till en funktion, så ska man göra det. Eftersom en sådan funktion kan anropas med en konstant och en vanlig variabel. Medan däremot en funktion, som har en vanlig pekare som argument, inte kan anropas med en konstant. En ytterligare förutsättning, för att detta ska vara möjligt, är att man inte ändrar på objekt eller värde i funktionen via pekaren.

När det gäller större objekt, så brukar man använda referenser snarare än pekare. En orsak till detta är att det förenklar koden inuti funktionen, eftersom man kan arbeta med argumenten som vanliga variabler. Mer om referenser längre fram.

Pekare som argument till funktioner (Värdeanrop referensanrop)

I avsnittet om funktioner såg vi att man kan skicka med värden till funktioner i form av funktionsparametrar. När det gäller enkla datatyper (vanliga) variabler, så är det endast deras värden man skickar som argument. Låt mig åskådliggöra detta med ett litet exempel.

argument_one.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;

void skriv(int nummer);

int main()
{
int nummer=5;
    skriv(8);
    skriv(nummer);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skriv(int nummer)
{
    cout<<"nummer="<<nummer<<endl;
}
```

Vi har en enkel funktion, skriv, som tar ett heltal, som argument och skriver ut dess värde på skärmen. Föga förvånande, så ger programmet följande utskrift:

```
nummer=8
nummer=5
Tryck på en valfri tangent för att fortsätta...
```

Vid anrop nummer ett, så skickar vi en literal, en vanlig siffra, som argument till funktionen. Vid anrop nummer två, så skickar vi med en variabel, men det är just det som vi inte gör? Vi skickar faktiskt bara med det värde, som nummer har.

Svårt att förstå kanske? Men jag ska visa vad jag menar med ett exempelprogram.

argument_two.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

void changeNumber(int nummer);

int main()
{
dos_console();
int nummer=5;
    cout<<"Värde på nummer i main = "<<nummer<<endl;
    changeNumber(nummer);
    cout<<"Ett anrop till changeNumber har skett."<<endl;
    cout<<"Värde på nummer i main = "<<nummer<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}

void changeNumber(int nummer)
{
    cout<<"Värde på nummer i changeNumber ="<<nummer<<endl;
    nummer+=10;
    cout<<"Värde på nummer i changeNumber ="<<nummer<<endl;
}
```

Programmet ger följande utskrift:

```
Värde på nummer i main = 5
```

```
Värde på nummer i changeNumber =5

Värde på nummer i changeNumber =15

Ett anrop till changeNumber har skett.

Värde på nummer i main = 5

Tryck på en valfri tangent för att fortsätta...
```

Om man studerar utskriften från programmet noga, så ser man att variabeln nummer i mainfunktionen inte är samma variabel, som i funktionen changeNumber. Det bara råkar vara så, i det här programmet att variablerna har samma namn. Uttrycket changeNumber(nummer) innebär inte, att vi skickar iväg själva variabeln nummer, utan endast dess värde, dvs. 5. Vi hade uppnått precis samma effekt genom att skriva changeNumber(5).

Men ibland, så har man behov av att låta den anropade funktionen faktiskt förändra värdet på argumentet, så att det så att säga får genomslag även vid anropsstället. Det är någonting man kan åstadkomma med pekare. Vi ska byta ut argumentet i funktionen changeNumber till en pekare till int. I anropet, så skickar vid med pekaradressen till nummer.

argument_three.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

void changeNumber(int* nummer);

int main()
{
dos_console();
int nummer=5;
    cout<<"Värde på nummer i main = "<<nummer<<endl;
    changeNumber(&nummer);
    cout<<"Ett anrop till changeNumber har skett."<<endl;
    cout<<"Värde på nummer i main = "<<nummer<<endl;
```

```
    system("PAUSE");
    return EXIT_SUCCESS;
}


void changeNumber(int* nummer)
{
    cout<<"Värde på nummer i changeNumber ="<<*nummer<<endl;
    *nummer+=10;
    cout<<"Värde på nummer i changeNumber ="<<*nummer<<endl;
}
```

Observera att jag förenklat utskrifterna lite. Istället för att skriva, ”Värdet på den adress, som nummer pekar på”, så har jag skrivit ”Värde på nummer i main osv.”.

Studera utskriften från programmet nedan:

```
Värde på nummer i main = 5
Värde på nummer i changeNumber =5
Värde på nummer i changeNumber =15
Ett anrop till changeNumber har skett.
Värde på nummer i main = 15
Tryck på en valfri tangent för att fortsätta...
```

Jo, nu såg det allt lite annorlunda ut. Nu är det inte bara ett värde vi skickar som argument till funktionen, utan variabeln nummers primärminnesadress. Man skulle förenklat uttryckt kunna säga, att vi skickade själva variabeln.

Med hjälp av pekare, som argument till en funktion, så kan man få funktionen att returnera mer än ett värde.

Man kan åstadkomma detta också med ytterligare en variabeltyp, nämligen referenser. En slags konstant pekare till en variabel. VI ska kika på referenser lite längre fram i boken.

Man skulle till exempel kunna tänka sig en funktion, som tar en pekare till int som argument. I funktionen räknas det värde, som pekare pekar på, upp med ett. Sedan kollas, om det nya värdet är jämnt delbart med t.ex. 5, då sant returneras, annars så returneras falskt. I ett anropande program, kanske man, av någon anledning måste arbeta med tal jämnt delbara med 5.

even_to_five.cpp

```
#include <cstdlib>
```

```
#include <iostream>
#include<iodos.h>

using namespace std;

bool evenToFive(int* nummer);

int main()
{
dos_console();
int nummer=0;

    while(nummer<100)
        if(evenToFive(&nummer))
        cout<<nummer<<" är jämnt delbart med 5"<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}

bool evenToFive(int* nummer)
{
   (*nummer)++; /*parentesen beror på
                  att tecknet * är ett
                  pekartecken och inte
                  en matematisk operator
                  denna gång. Tar man bort
                  parentesen, så fungerar
                  programmet felaktigt. Vi ska
                  längre fram kika på något, som
                  kallas för pekararitmetik.
```

```
                    */


   if(*nummer % 5 == 0)
        return true;
    return false;


}
```

En förkortad utskrift från programmet ses nedan.

```
5 är jämnt delbart med 5
10 är jämnt delbart med 5
15 är jämnt delbart med 5
20 är jämnt delbart med 5
25 är jämnt delbart med 5
30 är jämnt delbart med 5
35 är jämnt delbart med 5
40 är jämnt delbart med 5
45 är jämnt delbart med 5
50 är jämnt delbart med 5
Tryck på en valfri tangent för att fortsätta...
```

Det här är väl förmodligen inte det mest användbara program som skapats, men det illustrerar i alla fall, hur man kan använda möjligheten att låta en funktion ha mer än ett returvärde.

Swapping – att skifta värden

Vi ska kika på ytterligare ett exempel, som illustrerar skillnaden mellan värdeanrop och pekaranrop till en funktion. Det här exemplet brukar återkomma i programmeringslitteraturen, men eftersom jag inte vet vem som var först, så ber jag att få tacka alla som använt exemplet i sina böcker. Det är nämligen väldigt bra!

Låt oss anta, att vi har två variabler och vill skifta värden på dessa.

```
int a = 10;
int b = 5;
```

Vi vill alltså att a ska få värdet 5 och b värdet 10. Då kan man använda en teknik, som i programmeringssammanhang brukar kallas för swapping. Jag vågar utlova att du kommer att få se den tillämpad fler gånger i denna bok, bl.a. när vi ska sortera data längre fram.

För att vi ska kunna byta värden på a och b, så kan vi inte säga:

```
a=b;  //Fungerar
b=a; //Fungerar inte, eftersom a har b:s gamla värde.
```

Vi måste helt enkelt blanda in ytterligare en variabel. I det här fallet av heltalstyp.

```
int a = 10;
int b = 5;
int temp = a; //Tillfällig mellanlagring
a=b; //Fungerar
b = temp; //Nu får b a:s gamla värde
```

Man skulle kunna göra en funktion, som gör jobbet. Vi kan kalla den för swap.

swap_one.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;

void swap(int a,int b);
int main()
{
int a = 10;
int b = 5;
    cout<<"a="<<a<<" b="<<b<<endl;
    swap(a,b);
    cout<<"a="<<a<<" b="<<b<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
void swap(int a,int b)
{
    int temp=a;
    a=b;
```

```
    b=temp;
    cout<<"I swap a="<<a<<" b="<<b<<endl;
}
```

En provkörning av programmet gav följande utskrift:

```
a=10 b=5
I swap a=5 b=10
a=10 b=5
Tryck på en valfri tangent för att fortsätta...
```

Programmets utskrift visar på ett tydligt sätt att det hela inte fungerade som tänkt. Och jag tror att du redan vet orsaken. Vi har arbetat med värdeanrop. Att a och b inuti funktionen har bytt plats gör ingen som helst nytta i mainfunktionen, där funktionen anropas. Det beror, som sagts tidigare, på att variablerna i funktionen swap inte är desamma, som de i mainfunktionen, namnen till trots. Det finns två sätt att lösa problemet på. Det första kan du redan. Använd pekarargument. Det andra ska vi kika på strax. Använd referensargument.

Vi skriver om funktionen, så att den tar pekare till heltal, argument.

swap_two.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;

void swap(int* a,int* b);

int main()
{
int a = 10;
int b = 5;
    cout<<"a="<<a<<" b="<<b<<endl;
    swap(&a,&b);//Skicka adresserna istället.
    cout<<"a="<<a<<" b="<<b<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
void swap(int* a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
    cout<<"I swap a="<<*a<<" b="<<*b<<endl;
}
```

En provkörning av programmet visar, att det fungerar, som det ska.

```
a=10 b=5
I swap a=5 b=10
a=5 b=10
Tryck på en valfri tangent för att fortsätta...
```

Ovan sa jag ju, att det fanns ytterligare ett sätt, nämligen att använda referenser. Det ska vi titta lite närmare på nu.

Referenser och referenser som argument till funktioner

I C++ finns en språkkonstruktion, som påminner väldigt mycket om pekare, nämligen referensen. Det är därför väldigt lätt att man blandarihop referenser och pekare.

Man skulle kunna säga att en referens är en konstant pekare till en viss variabel. Det innebär att man alltid måste initiera en referens. Det finns några undantag från denna regel . Ett av dessa undantag är, när man använder referenser som parametrar (argument) till funktioner.

Vi kikar förstås på lite kod.

ref_one.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int tal=45;
//int &ref; Kompileringsfel. Referensen måste initieras
/*Observera att & inte är en adressoperator denna gång
  , utan innebär att ref är en referens till tal.*/
```

```
int &ref=tal;
        cout<<"ref: "<<ref<<endl;
        ref+=10;
        cout<<"tal: "<<tal<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En utskrift av programmet ser ut så här:

```
ref: 45
tal: 55
Tryck på en valfri tangent för att fortsätta...
```

Programmet visar tydligt att vi via referensen ref kunde ändra värdet på tal. Det är helt enkelt så, att ref är ett alias för tal. Vi ska nu göra om programmet, som bytte värden på två variabler och använda referensargument istället för pekare.

swap_med_ref.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
void swap(int& a,int& b);
int main()
{
int a=5;
int b=10;
    cout<<"a="<<a<<" b="<<b<<endl;
    swap(a,b);
    cout<<"a="<<a<<" b="<<b<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
void swap(int& a,int& b)
```

```
{
    int temp=a;
    a=b;
    b=temp;
    cout<<"I swap a="<<a<<" b="<<b<<endl;
}
```

Programmet gav följande utskrift:

```
a=5 b=10
I swap a=10 b=5
a=10 b=5
Tryck på en valfri tangent för att fortsätta...
```

Utskriften visat att programmet fungerade på precis samma sätt, som när vi använde pekare som parametrar. Skillnaderna står att finna i koden. När vi anropade swap den här gången, så behövde vi inte skicka med minnesadresserna. Funktionen swap, ser väl betydligt enklare ut den här gången. Vi behöver inte använda *, för att få fram det, som pekarna pekar på. För att påminna, om hur det såg ut, när vi använde pekare, så visar jag den lösningen en gång till.

```
void swap(int* a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
    cout<<"I swap a="<<*a<<" b="<<*b<<endl;
}
```

Vad ska man använda då, pekare eller referenser? Ja, för att svara på den frågan, så måste jag delvis gå utanför ramarna för den här boken. Om man ska skicka större objekt, som argument till en funktion, så ska man helst använda referenser. *Objekt har med objektorienterad programmering att göra, så det kommer vi inte att närmare in på i den här boken. Däremot, så kommer vi att använda strukturobjekt och då gäller ovanstående resonemang.*

Som vi såg i avsnittet om pekare, så skulle man så långt möjligt använda sig av konstanta pekare, som argument i funktionernas parameterlistor, eftersom dessa kan anropas med både vanliga pekare och konstanta pekare. Medan vanliga pekare i funktionens parameterlista bara kan anropas med vanliga pekare som argument. Men för att kunna använda konstanta pekare i en parameterlista fordras dessutom att man inte utför någon förändring av pekaren eller det värde den pekar på i funktionen, annars får man kompileringsfel.

Så en uppdelning skulle kunna vara att man använder referenser som argument, om man behöver utföra förändringar av argumentens värde i funktionen, annars använder man konstanta pekare.

Mina personliga erfarenheter säger mig, att referenser är att föredra, eftersom de är lättare att hantera. Men det är inte alltid, som man kan använda referenser.

Pekare och arrayer – samma sak

Jo, så är det, pekare och arrayer är precis samma sak. Egentligen handlar det om att arrayer utnyttjar någonting, som kallas för pekararitmetik. Pekararitmetik ska vi gå igenom lite längre fram, men kort innebär det, att man stegar i primärminnet. En array är ett antal värden, som ligger på rad. Man kan stega sig igenom arrayer och i övrigt hämta värden från och tilldela värden till arrayens olika element via indexering.

Arrayer finns i många andra programmeringsspråk, men i C++ är arrayens inre funktionalitet väldigt tydlig. Skulle jag rekommenderare utövare av andra programmeringsspråk att lära sig något i C++, så är det just, arrayer, pekare och pekararitmetik, eftersom det är så oerhört lärorikt.

Vi ska väl först pröva påståendet, att pekare och arrayer är utbytbara storheter. När man skapar en funktion, som ska bearbeta en array, som den fått som parameter, så borde man alltså kunna byta ut arrayparametern mot en pekare. Vi gör ett försök.

```
pek_one.cpp
#include <cstdlib>

#include <iostream>


using namespace std;

/* OBS första argumentet till funktionen är inte

   en array, utan en pekare till int.*/

void skrivArray(int* array,int storlek);


int main()

{

int tal[]={5,11,23};

int storlek = sizeof(tal)/sizeof(int);

skrivArray(tal,storlek); //Vi skickar en array som argument

    system("PAUSE");

    return EXIT_SUCCESS;

}
```

```
/* OBS första argumentet till funktionen är inte
   en array, utan en pekare till int.*/


void skrivArray(int* array,int storlek)
{
    for(int i=0;i<storlek;i++)
  /* Lägg märke till att arrayindexering används
     på pekarargumentet */
        cout<<"array["<<i<<"]="<<array[i]<<endl;
}
```

När man skickar en array som argument till en funktion, som ska ha en pekare som argument, så sker en automatisk typomvandlig (Se avsnittet Typkonverteringar). I funktionen, så kan man sedan använda arrayindexering, när man vill komma åt enskilda element i argumentet. Det ser naturligtvis jätteskumt ut, men beror egentligen på att pekare och arrayer i grund och botten är samma sak. Vi har tidigare i den här boken, skickat arrayer som argument till funktioner. Men det är faktiskt så, att vi egentligen inte har skickat någon array som argument, utan en pekare, dvs. adressen till arrayens första element.

Det är det som gör att vi måste skicka med arrayens storlek till funktionen. Storleken går nämligen inte att beräkna i funktionen, för den känner bara till första elementet i arrayen. Jag måste naturligtvis bevisa mitt påstående med ett program.

array_funk_test.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
void skrivArray(int tal[]);
int main()
{
int tal[]={67,23,12};
skrivArray(tal);
    system("PAUSE");
    return EXIT_SUCCESS;
}
void skrivArray(int tal[])
```

```
{
int storlek = sizeof(tal)/sizeof(int);
cout<<"storlek ="<<storlek<<endl;
    for(int i=0;i<storlek;i++)
        cout<<"tal["<<i<<"]="<<tal[i]<<endl;
}
```

En provkörning av programmet gav följande alarmerande utskrift:

```
storlek =1
tal[0]=67
Tryck på en valfri tangent för att fortsätta...
```

Som du kan se, så är arrayens storlek 1. Men tittar vi efter i mainfunktionen, så kan vi se att det finns tre element i arrayen. Men nu var det ju inte en array vi skickade, utan adressen till arrayens första element och där ligger ju förvisso talet 67. Det är också det enda värdet, som skriv ut i for-loopen.

Pröva gärna att skriva om programmet genom att byta ut argumentet i funktionens parameterlista till en pekare till int.

```
void skrivArray(int* tal); //Prototyp


void skrivArray(int* tal)
{
int storlek = sizeof(tal)/sizeof(int);
cout<<"storlek ="<<storlek<<endl;
    for(int i=0;i<storlek;i++)
        cout<<"tal["<<i<<"]="<<tal[i]<<endl;
}
```

Om du testar programmet, så kommer du inte att se någon skillnad i programmets utskrift.

Pekararitmetik

Pekararitmetik är till att börja med ett ganska besvärligt ord, men låt dig inte avskräckas. Det betyder helt enkelt, att man kan utföra matematiska operationer på själva pekaren i sig. Att man kan utföra matematiska operationer på det värde, som pekaren pekar på, är självklart, om det är ett tal av något slag som ligger där. Men vad menas med matematiska operationer på pekaren i sig?

Jo med det menas, att man kan förflytta pekaren till en ny minnesadress, t.ex. genom att öka pekaren med 1. Då kommer pekaren att flytta på sig ett steg i minnet. Stegets storlek bestäms av storleken på den datatyp, som pekaren pekar på. Vi ska kika på några exempel.

aritmetik_one.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;

void skrivData(int* pek,int antal);

int main()
{
int numbers[]={5,11,23};
int storlek = sizeof(numbers)/sizeof(int);
skrivData(numbers,storlek);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivData(int* pek,int antal)
{
    for(int i=0;i<antal;i++)
    {
        cout<<"pek["<<i<<"]="<<*(pek+i)<<endl;//?
    }
}
```

Titta nu raden med kommentarstecknet i funktionen. Där står nu ett frågetecken. Vad i hela världen betyder nu *(pek+i)?

Funktionen tar en pekare till heltal som argument. Vi skickar en array till funktionen. Arrayen förvandlas till en pekare till int, genom automatiskt typkonvertering. Det innebär att pek är minnesadressen till arrayens första element. Det innebär, att om man ökar pekarens värde med ett, så pekar pekare på arrayens andra element. Vi har fått en förflyttning i minnet. Stegets storlek är lika stor, som storleken på en int. Med uttrycket *(pek+2) kommer vi åt det värde, som ligger på minnesadressen två steg från första minnesadressen i arrayen. Det betyder helt enkelt att pek[2] är precis samma sak som *(pek+2). Och med ens så förstår du precis, hur en array fungerar.

Programmet gav följande utskrift:

```
pek[0]=5

pek[1]=11

pek[2]=23

Tryck på en valfri tangent för att fortsätta...
```

Man kan utnyttja pekararitmetiken i många sammanhang. Ett sådant sammanhang är en for-loop.

aritmetik_two.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
void skrivData(int* pek,int antal);
int main()
{
int numbers[]={5,11,23};
int storlek = sizeof(numbers)/sizeof(int);
skrivData(numbers,storlek);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivData(int* pek,int antal)
{
    for(int *i=pek;i<pek+antal;i++)
    {
        cout<<*i<<endl;
    }
}
```

Studera for-loopen i skrivData. Där initieras nu en pekare döpt till i, till att peka på samma minnesadress som pek. I villkoret sägs att så länge, som i, vilket nu är en minneadress, är mindre än pek + antal, dvs. adressen för pek ökat med så många steg, som det finns platser i arrayen, så räknar vi upp i med 1, dvs. vi flyttar i ett steg i minnet.

programmet gav följande utskrift:

```
5

11

23

Tryck på en valfri tangent för att fortsätta...
```

Vi ska ta och kika på ett specialfall, när det gäller pekarargument till funktioner. Vi har nu sett i ett flertal exempel att vi måste skicka med antalet element i den medsända arrayen, för att kunna iterera den i t.ex. en for-loop. Vi vet också varför vi måste göra det. Orsaken är, att när vi skickar en array som argument, så skickar vi egentligen minneadressen till arrayens första element, så därför kan vi inte dynamiskt beräkna arrayens storlek i en funktion.

Det finns ett specialfall, nämligen den gamla csträngen. Som du kanske kommer ihåg, så definieras en csträng, som en array av char med ett avslutande nolltecken. Nolltecknet används till att avgöra, när strängen är slut. Detta borde innebära, att man skulle kunna slippa skicka med antalet tecken i en chararray, när man använder den som argument till en funktion.

Vi ska helt enkelt göra en egen version strlen, vilket är en funktion, som tar en pekare till char som argument och returnerar antalet tecken i strängen minus det avslutande nolltecknet. Vi kikar på lite kod först innan jag förklarar, hur det fungerar.

aritmetik_three.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;
int string_length(char*);

int main()
{
    dos_console();
    char str[]="Morgan";
    cout<<str<<" har "<<string_length(str)<<" tecken."<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

int string_length(char* s)
```

```
{
int l=0;
    for(char* i=s;*i;i++)
            l++;
        return l;
}
```

Studera nu for-loopen I funktionen string_length. Den börjar med att en lokal variabel l deklareras och tilldelas värdet 0. Sedan har vi for-loopen, som ser ut så här:

```
for(char* i=s;*i;i++)
```

En pekare till char döpt till i, ställs till att peka på samma adress, som argumentet s. Villkoret ser väl lite underligt ut, *i.

Eftersom sista tecknet i en csträng är 0, så kommer det, som i pekar på, förr eller senare att peka på värdet 0. Värdet 0 konverteras till bool med värdet false. De sker genom automatisk typkonvertering. *Se detta avsnitt i boken.* Det innebär enkelt uttryckt, att när det som i pekar på är en nolla, så kommer for-loopen att sluta exekvera. Sista avsnittet i for-loopen, i++, är en pekarstegning. Enligt principerna för pekararitmetik, så kommer i++ innebära, att pekaren flyttar på sig ett steg i primärminnet, där stegets längd är lika stort, som lagringsutrymmet för en char.

I for-loopen räknas variabeln l upp ett steg för varje varv i for-loopen. När for-loopen stannar av, så håller l värdet av antal tecken i argumentet, som ju är en csträng.

En testkörning av programmet gav följande resultat:

```
Morgan har 6 tecken.

Tryck på en valfri tangent för att fortsätta...
```

Att allokera minne dynamiskt med new– dynamiska arrayer

Vi har i några tidigare exempel i denna bok mött problemet, att vi inte på förhand kunnat säga, hur många platser vi har behövt i en array. Ett sätt att lösa problemet på, skulle kunna vara att man skapar en array med tillräckligt stort utrymme. Problemet med det är förstås, att om man i flertalet programkörningar endast använder en bråkdel av platserna, så slösar man med datorns resurser. Värre ändå, trots att man tagit i med platser, så man nästan spricker, så kommer programkörningen när platserna inte räcker i alla fall.

Lösningen är att dynamiskt allokera (använda/placera) minnesutrymme med nyckelordet new. I datorns finns ett flertal olika minnesutrymme:

- **Automatiskt minnesutrymme** – används vid hantering av vanliga variabler.
- **Dynamiskt minneutrymme** – kan man begära att få låna. Nyckelordet är new. Utrymmet måste lämnas tillbaka.

Till exempel, så kan man be att få låna utrymme i det dynamiska minnesutrymmet, till en pekare med uttrycket: int* pek = new int;

Eller varför inte till 5 st. int: int* pek = new int[5];

Jo, man kan begära att få låna. Det innebär inte att datorn måste bevilja begäran. Numer, så är datorerna som regel så väl utrustade med minnesutrymme, så det ska väl inte behöva hända att man blir nekad. Jag har själv aldrig råkat ut för det.

Lån innebär också att man måste lämna tillbaka. När man är färdig med ett dynamiskt allokerat minneutrymme, så lämnar man tillbaka det med delete. I exemplen ovan, så skulle det se ut på följande sätt:

```
delete pek;
delete [] pek; //[] eftersom pek var en array
```

Ja, då var det väl dags att kika på ett exempel igen då. Jag visar först lite kod. Jag förklarar efter koden.

pek_new_one.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;
int getNumber();

int main()
{
    dos_console();
    cout<<"Hur många tal vill du mata in?"<<endl;
    int antal = getNumber();
    int* tal = new int[antal];
    int start=0;
        while(start<antal)
        {
            cout<<"Ge mig tal nummer "<<(start+1)<<": ";
            tal[start]=getNumber();
            start++;
```

```
        }

        for(int i=0;i<antal;i++)
        {
            cout<<"tal["<<i<<"]="<<tal[i]<<endl;
        }

        delete[]tal;
    system("PAUSE");
    return EXIT_SUCCESS;
}
int getNumber()
{
    int retval = 0;
    while(!(cin>>retval))
    {
        cout<<"Ett heltal tack!"<<endl;
        cin.clear();
        cin.ignore(100,'\n');
    }
        return retval;
}
```

I det här programmet skapas en array av int och det är användaren, som avgör hur många heltal, som ska skapas och lagras i arrayen. Programmet använder en funktion, som returnerar ett heltal. Funktionen säkerställer att inga ogiltiga tal matas in av användaren. Minnesutrymmet allokeras dynamiskt med uttrycket: int* tal = new int[antal]. Observera att minneutrymmet lämnas tillbaka innan programmet avslutas med, delete[]tal.

Programmet gav följande utskrift vid en testkörning:

```
Hur många tal vill du mata in?
3
Ge mig tal nummer 1: 56
Ge mig tal nummer 2: fem
```

```
Ett heltal tack!

23

Ge mig tal nummer 3: 11

tal[0]=56

tal[1]=23

tal[2]=11

Tryck på en valfri tangent för att fortsätta...
```

En funktion, som returnerar en csträng, där dess längd och tecken anges som argument

I programmet hänga gubbe skapade vi en sträng med så många stjärnor, som datorns ord hade bokstäver. Det såg ut ungefär på följande sätt:

```
string stars="";
stars.append(antal_tecken,'*');
```

Vi ska nu skapa och använda en funktion, som tar antalet tecken och ett tecken som argument. Prototypen kommer att se ut så här:

```
char* makeString(int antal,char tecken);
```

Funktionen ska returnera en pekare till char. Eftersom man inte på förhand kan avgöra, hur lång strängen ska bli, så måste man allokera minne med new i funktionen. Utöver att vi inte på förhand kan avgöra, hur många tecken, som strängen ska innehålla, så kan vi inte returnera an vanlig array. Studera koden nedan:

```
char* makeString(int antal,char tecken)
{
char retval[1000];
          for(int i=0;i<antal;i++)
          {
              retval[i]=tecken;
              retval[i+1]='\0';
          }
             return retval;
}
```

Det är väl ganska troligt, att 1000 tecken räcker till de allra flesta strängar. Vi avslutar dessutom strängen genom att lägga ett 0-tecken längst bak i arrayen. Men funktionen lider av en annan felaktighet. Eftersom retval skapas inuti funktionen, så upphör den att existera, så fort exekveringen

av funktionen upphör. Det innebär att man returnerar en pekare till en adress, som inte längre är reserverat för retval. Det finns en ganska stor risk för att data, som ligger på minneadressen, skrivs över någon annanstans i programmet.

En korrekt version, ser ut på följande sätt:

```
char* makeString(int antal,char tecken)
{
char* temp = new char[antal+1];
          for(int i=0;i<antal;i++)
          {
              temp[i]=tecken;
              temp[i+1]='\0';
          }
             return temp;
}
```

Rätt modell är att allokera minne med new. Då används nämligen det dynamiska minnesutrymmet. Det som ligger lagrat där kan bara tas bort av dig. Det finns således ingen risk att det skrivs över under programmets livstid. Men och återigen men glöm inte bort att frigöra minnet. Vi tittar väl på ett litet exempel, där vi använder funktionen.

make_string_testare.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;
char* makeString(int antal,char tecken);

int main()
{
    dos_console();
    char tecken[]="?*<>|^";

    for(int i=0;i<strlen(tecken);i++)
    {
```

```
        char* str = makeString(i+10,tecken[i]);
        cout<<str<<endl;
        delete [] str; //OBS Glöm ej!
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
char* makeString(int antal,char tecken)
{
/* Vi allokerar minne med new. Utrymmet
   vi allokerar är lika med antal+1, eftersom
   vi måste ha plats för det avslutande 0-tecknet.
*/
char* temp = new char[antal+1];
          for(int i=0;i<antal;i++)
          {
              temp[i]=tecken;
              temp[i+1]='\0';
          }
             return temp;
}
```

Programmet gav följande utskrift:

```
??????????
**********
<<<<<<<<<<<<
>>>>>>>>>>>>>>
||||||||||||
^^^^^^^^^^^^^^^
Tryck på en valfri tangent för att fortsätta...
```

Pekare till funktioner

Jo, visst kan man skicka även funktioner som argument till andra funktioner. Varför skulle man vilja det? Ja, i vissa situationer, så kan det vara praktiskt att skicka med en pekare till en funktion, som argument till en funktion, eftersom funktionen behöver anropa olika andra funktioner från tillfälle till tillfälle, beroende på omständigheterna.

Att hämta adressen till en funktion är relativt lätt. Det är helt enkelt funktionens namn utan parentesen. Adressen till skriv() är skriv.

Att deklarera en pekare till en funktion set lite krångligare ut. Men man kan skriva funktionen precis som vanlig: void skriv(). Sedan byter man ut funktionens namn mot t.ex. (*pf) , vilket skulle ge:

```
void (*pf) ();
```

En funktionsprototyp som ska ha en pekare till en funktion, som returnerar void och inte ska ha några argument som argument ser ut på följande sätt:

```
void geInformation(void (*pf)());
```

Vi kikar väl på ett lite enklare exempel.

funk_pek_one.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;
void skriv();
void geInfo(void (*pf)());
int main()
{
    dos_console();
    geInfo(skriv);//Skickar adressen till skriv som argument
    system("PAUSE");
    return EXIT_SUCCESS;
}
void skriv()
{
    cout<<"skriv ger information"<<endl;
}
```

```
void geInfo(void (*pf)())
{
    cout<<"geInfo anropar pf"<<endl;
    pf();//Anropar funktionen, som kom som argument
}
```

Programmet gav följande utskrift:

```
geInfo anropar pf
skriv ger information
Tryck på en valfri tangent för att fortsätta...
```

Ett praktiskt exempel skulle väl kunna vara, att en pekare till en funktion under vissa omständigheters pekar på en funktion och under andra omständigheter på en annan. Här nedan ges koden till ett program, som beroende på om användaren är engelsktalande eller svensktalande ger ett välkomstmeddelande till en bank på engelska eller svenska.

Det finns en funktion, som heter svenska och som returnerar strängen, ”Välkommen till Banken”. En annan funktion heter engelska och returnerar strängen, ”Welcome to the Bank”.

I programmet (mainfunktionen) finns en pekare till en funktion, som returnerar en pekare till char. Användare får frågan på engelska, ”Do you speak english? [y-n]”. Om användare skriver no, så ställs funktionspekaren till att peka på funktionen svenska. Alla andra alternativ ger som resultat, att funktionspekaren pekar på english. En funktion, som heter skrivMeddelande tar som argument en funktionspekare. Eftersom funktionspekaren pekar till en funktion, som returnerar en pekare till char, så används funktionspekaren till att hämta rätt sträng, som skrivs ut på skärmen.

languages.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>

using namespace std;

char* svenska();
char* engelska();
void skrivMeddelande(char*(*pf)());
```

```
int main()
{
    dos_console();
    char* (*pf)();
    char c='y';
    char in;
    cout<<"Do you speak english? [y-n]"<<endl;
    cin.get(in);
        if(in=='n')
            c='n';
    if(c=='n')
        pf = svenska;
    else
        pf = engelska;
    skrivMeddelande(pf);

    system("PAUSE");
    return EXIT_SUCCESS;
}

void skrivMeddelande(char*(*pf)())
{
    char* m = pf();
    cout<<m<<endl;
    delete [] m;
}

char* svenska()
{
char lokal[]="Välkommen till Banken";
char* temp = new char[strlen(lokal)+1];
```

```
strcpy(temp,lokal);
    return temp;
}
char* engelska()
{
char locale[]="Welcome to the Bank";
char* temp = new char[strlen(locale)+1];
strcpy(temp,locale);
    return temp;
}
```

Några testkörningar på min dator gav följande resultat:

```
Do you speak english? [y-n]
n
Välkommen till Banken
Tryck på en valfri tangent för att fortsätta...
```

```
Do you speak english? [y-n]
y
Welcome to the Bank
Tryck på en valfri tangent för att fortsätta...
```

Att sortera data – sorteringsalgoritmer

Att sortera data hör till programmeringens finrum. Mycket tankeköda och arbete har lagts ner bland programmerare världen över genom åren på att skapa så effektiva sorteringsalgoritmer som möjligt.

Tyvärr, så är det också ganska så svårt för nybörjare att ta till sig sorteringsalgoritmer, men jag lovar att göra mitt bästa, för att introducera dig på området.

Som sagt, det finns många sorteringsalgoritmer, bubblesort, quicksort och bidirectional sort osv. Många av dessa är oerhört effektiva, men också väldigt komplexa och svåra att förstå. När du blivit lite mer van programmerare och vill fördjupa dig i sorteringsalgoritmer, så finns rikligt med material på Internet. Naturligtvis återkommer sorteringsalgoritmer i programmeringslitteraturen.

Det är två begrepp, som är centrala i alla sorteringsalgoritmer:

Arrayer – Listor

All sortering av data sker i olika typer av listor. En array är också en lista.

Swap

Det är en teknik, som används för att två variabler ska kunna byta värden med varandra, utan att tappa data. Swapping har behandlats tidigare i denna bok, då jag skrev om pekare.

Jag kommer att fokusera på två sorteringsalgoritmer. En som jag kallar för naiv sortering och en som brukar kallas för bubbelsortering.

Naiv Sortering

Naiv sortering är kanske den allra enklaste sorteringsalgoritmen. Det är därför jag börjar med den. Den är en aning ineffektiv, om det är stora mängder med data som ska sorteras, men den duger till de mest vardagliga sorteringarna.

Låt oss anta att vi har antal heltal i en array. Om man visar det med en bild, så skulle det kunna se ut som nedan:

0	1	2	3	4
6	5	3	7	1

I den övre raden ligger adresserna. Dessa kommer aldrig att ändras, men vi skulle vilja ha värdena i ordningen, från minsta talet till det högsta. Då skulle man kunna göra så här.

Adress 0 jämför sitt värde med adress 1.

0	1	2	3	4
6	5	3	7	1

Eftersom 5 är lägre än 6, så skiftar adress 0 och adress 1 värden.

0	1	2	3	4
5	6	3	7	1

Sedan jämför adress 0 sitt värde med adress 2.

0	1	2	3	4
5	6	3	7	1

Eftersom 5 är högre än 3, så skiftar adress 0 värde med adress 2

0	1	2	3	4
3	6	5	7	1

Sedan jämför adress 0 sitt värde med adress 3

0	1	2	3	4
3	6	5	7	1

Eftersom 3 är lägre än 7, så sker ingenting.

Sedan så jämför adress 0 sitt värde med värdet på adress 4.

0	1	2	3	4
3	6	5	7	1

Eftersom 1 är lägre än 3, så byter adress 0 och 4 värden.

0	1	2	3	4
1	6	5	7	3

Nu har adress 0 jämfört sitt värde med alla andra adressers värden. Det betyder, att nu ligger arrayens lägsta värde på adress 0. Nu har turen kommit till adress 1.

Adress 1 jämför sitt värde med värdet på adress 2

0	1	2	3	4
1	6	5	7	3

Eftersom värdet på adress 2 (5) är lägre än värdet på adress 1(6), så skiftar adress 1 och 2 värden.

0	1	2	3	4
1	5	6	7	3

Sedan jämför adress 1 sitt värde med värdet på adress 3.

0	1	2	3	4
1	5	6	7	3

Eftersom värdet på adress 1(5) är lägre än värdet på adress 3(7), så sker ingenting.

Sedan jämför adress 1 sitt värde med värdet på adress 4

0	1	2	3	4
1	5	6	7	3

Eftersom värdet på adress 4(3) är lägre än värdet på adress 1(5), så skiftar dessa värden.

0	1	2	3	4
1	3	6	7	5

Nu har adress 1 gjort sitt. Arrayens näst lägsta värde ligger på adress 1.

Nu har turen kommit till adress 2 att göra sitt jobb.

Adress 2 jämför värdet på sin adress med värdet på adress 3

0	1	2	3	4
1	3	6	7	5

Eftersom värdet på adress 2(6) är lägre än värdet på adress 3, så sker ingenting.

Sedan jämför adress 2 sitt värde med värdet på adress 4

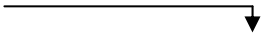
0	1	2	3	4
1	3	6	7	5

Eftersom värdet på adress 4(5) är lägre än värdet på adress 2(6), så skiftar dessa adresser värden.

0	1	2	3	4
1	3	5	7	6

Nu har adress 2 gjort sitt och arrayens tredje lägsta värde ligger på adress2.

Nu återstår det för adress 3 att göra sitt jobb.

Adress 3 jämför sitt värde med värdet på adress 4 

0	1	2	3	4
1	3	5	7	6

Eftersom värdet på adress 4 är lägre än värdet på adress 3(7), så skiftar adresserna värden.

0	1	2	3	4
1	3	5	6	7

Ja, då har adress 3 gjort sitt jobb och eftersom adress 4 inte har någon högre adress att jämföra med, så är sorteringsarbetet slutfört. En kontroll av värdena visar att sorteringen har fungerat.

Hur ska man nu skriva kod, som gör samma sak. En idé är förstås att skapa en for-loop och i denna for-loop lägga en inre for-loop, som hela tiden loopar från den yttre loopens index plus 1. Då får man som resultat, att den yttre loopens adress jämförs med alla adressers värden, som ligger till höger om den, på samma sätt som på bilderna ovan. Om man vid jämförelsen mellan adressernas värden finner att dessa ligger fel, så låter man adresserna skifta värden med hjälp av en swap.

Det har blivit dags för lite kod igen. Jag kommer att skapa en funktion, som tar en pekare till heltal som argument samt ett heltal, som anger hur stor arrayen är. Jag kommer för tydlighetens skull att använda precis samma array, som i bilderna ovan.

naiv_sort_one.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
void skrivArray(int* arr,int storlek);
void naivSortering(int* arr,int storlek);
int main()
{
    int tal[]={6,5,3,7,1};
    int storlek = sizeof(tal)/sizeof(int);
    cout<<"Osorterad array"<<endl;
    skrivArray(tal,storlek);
    naivSortering(tal,storlek);
    cout<<"Sorterad array"<<endl;
    skrivArray(tal,storlek);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
void skrivArray(int* arr,int storlek)
{
    for(int i=0;i<storlek;i++)
    {
        cout<<"arr["<<i<<"]="<<arr[i]<<endl;
    }
}
void naivSortering(int* arr,int storlek)
{
int temp=0; //Används till swappen
    for(int i=0;i<storlek;i++)
    {
        for(int j=i+1;j<storlek;j++)
        {
            if(arr[j]<arr[i])
            {
                /*Här swappas det*/
                temp = arr[j];
                arr[j]=arr[i];
                arr[i]=temp;
            }
        }
    }
}
```

Programmet gav följande utskrift:

```
Osorterad array
arr[0]=6
arr[1]=5
arr[2]=3
arr[3]=7
arr[4]=1
```

```
Sorterad array

arr[0]=1

arr[1]=3

arr[2]=5

arr[3]=6

arr[4]=7

Tryck på en valfri tangent för att fortsätta
```

Jag tänkte att vi skulle lägga till några funktioner, så att man lättare kan följa sorteringsarbetet på skärmen. Vad jag vill åstadkomma är utskrifter enligt nedan:

```
arr[0]->2       --

arr[1]->11      ----------

arr[2]->3       ---

arr[3]->1       -

arr[4]->7       -------

arr[5]->8       --------

*************************

arr[0] och arr[3]skiftar värdena 2 och 1

arr[0]->1       -

arr[1]->11      ----------

arr[2]->3       ---

arr[3]->2       --

arr[4]->7       -------

arr[5]->8       --------

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

Vi behöver ha en ny funktion, som skriver ut en arrays innehåll enligt ovan. Vi skulle kunna kalla den för skrivFormateradArray. Vi skulle också ha nytta av en funktion, som returnerar en sträng med bindestreck. Bäst är det väl, om man anger, hur många streck strängen ska bestå av med ett argument till funktionen. Vi skulle kunna kalla funktionen för buildString. Sist men inte minst, så skriver vi en ny sorteringsfunktion. Vi kallar den för naivSortering. Vi ska använda naiv sortering, dvs. den algoritm, som du kan läsa om ovan, men vi ska lägga in några beskrivande utskrifter under

sorteringsarbetet. En av fördelarna med den här programdesignen är att vi skulle kunna lägga in funktioner med andra sorteringsalgoritmer.

Vi börjar med headerfilen.

sortering.h

```
#ifndef _SORTERING_H
#define _SORTERING_H
#include<iostream>
using namespace std;
void  naivSortering(int*,int antal);
void  skrivFormateradArray(int*,int antal);
char* buildString(int antal,char c='-');
#endif
```

Det var väl inga konstigheter här tror jag. Innehållet följer den planering, som vi gjorde ovan. Då ska vi kika på källkodsfilen. Jag tror du förstår, hur funktionerna är implementerade. Vi har gått igenom det mesta av det tidigare i boken. Om du tittar i funktionen naivSortering, så kan du se, att den ser precis likadan som förut. Den enda skillnaden är att jag har lagt in lite klargörande utskrifter.

sortering.cpp

#include "sortering.h"

```
void naivSortering(int* arr,int antal)
{
int temp=0;
    for(int i=0;i<antal;i++)
    {
        for(int j=i+1;j<antal;j++)
        {
            if(arr[i]>arr[j])
            {
                cout<<"arr["<<i<<"] och arr["<<j<<"]skiftar ";
                cout<<"värdena "<<arr[i]<<" och "<<arr[j]<<endl;
                temp = arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
```

```
                skrivFormateradArray(arr,antal);
            }


        }
    }
}
void skrivFormateradArray(int* arr,int antal)
{
    for(int i=0;i<antal;i++)
    {
        char* arr_str = buildString(arr[i]);
        cout<<"arr["<<i<<"]->"<<arr[i]<<"\t"<<arr_str<<endl;
        delete [] arr_str;
    }
    cout<<"*************************"<<endl;
}


char* buildString(int antal,char c)
{
    char* temp=new char[antal+1];
    for(int i=0;i<antal;i++)
    {
        temp[i]=c;
        temp[i+1]='\0';
    }
    return temp;
}
```

Slutligen, så ska vi väl testa funktionerna I ett program. Programmet fungerar ungefär som föregående program. Enda skillnaden är egentligen, att vi nu anropar funktionen skrivFormateradArray istället för skrivArray.

naiv_sort_two.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include "sortering.h"
using namespace std;
int main()
{
    dos_console();
int data[]={6,5,3,7,1};
int storlek = sizeof(data)/sizeof(int);
skrivFormateradArray(data,storlek);
naivSortering(data,storlek);
          system("PAUSE");
    return EXIT_SUCCESS;
}
```

En programkörning gav följande resultat:

```
arr[0]->6     ------
arr[1]->5     -----
arr[2]->3     ---
arr[3]->7     -------
arr[4]->1     -
*************************
arr[0] och arr[1]skiftar värdena 6 och 5
arr[0]->5     -----
arr[1]->6     ------
arr[2]->3     ---
arr[3]->7     -------
arr[4]->1     -
*************************
arr[0] och arr[2]skiftar värdena 5 och 3
```

```
arr[0]->3       ---

arr[1]->6       ------

arr[2]->5       -----

arr[3]->7       -------

arr[4]->1       -

*************************

arr[0] och arr[4]skiftar värdena 3 och 1

arr[0]->1       -

arr[1]->6       ------

arr[2]->5       -----

arr[3]->7       -------

arr[4]->3       ---

*************************

arr[1] och arr[2]skiftar värdena 6 och 5

arr[0]->1       -

arr[1]->5       -----

arr[2]->6       ------

arr[3]->7       -------

arr[4]->3       ---

*************************

arr[1] och arr[4]skiftar värdena 5 och 3

arr[0]->1       -

arr[1]->3       ---

arr[2]->6       ------

arr[3]->7       -------

arr[4]->5       -----

*************************

arr[2] och arr[4]skiftar värdena 6 och 5

arr[0]->1       -
```

```
arr[1]->3       ---

arr[2]->5       -----

arr[3]->7       -------

arr[4]->6       ------

*************************

arr[3] och arr[4]skiftar värdena 7 och 6

arr[0]->1       -

arr[1]->3       ---

arr[2]->5       -----

arr[3]->6       ------

arr[4]->7       -------

*************************

Tryck på en valfri tangent för att fortsätta…
```

Det är säkert väldigt lärorikt att studera utskrifterna ovan. Nu ska vi ge oss i kast med ytterligare en sorteringsalgoritm, bubbelsortering.

Bubbelsortering

Ett härligt namn på en sorteringsalgoritm, eller hur? Den har fått sitt namn efter den rörelse som arrayens värden företar under algoritmens gång. Det ser nämligen ut som om värdena bubblar rätt.

I en bubbelsortering kan också med fördel använda två nästlade for-loopar. Men den yttre for-loopen har en helt annan roll i bubbelsorteringen. Den ser egentligen bara till att arrayen scannas igenom lika många gånger, som det finns element i arrayen. Varje scanning ser ut enligt nedan:

0	1	2	3	4
6	5	3	7	1

Adresserna 0 och 1 jämför sina värden. Eftersom värdet på adress 1 är lägre än värdet på adress 0, så skiftas värdena.

0	1	2	3	4
5	6	3	7	1

0	1	2	3	4
5	6	3	7	1

Adresserna 1 och 2 jämför värdena på sina adresser. Eftersom 3 är lägre än 6, så skiftas värdena.

0	1	2	3	4
5	3	6	7	1

0	1	2	3	4
5	3	6	7	1

Adresserna 2 och 3 jämför värdena på sina adresser. Eftersom 6 är lägre än 7, så sker inte något skifte.

0	1	2	3	4
5	3	6	7	1

Adresserna 3 och 4 jämför sina värden. Eftersom 1 är lägre än 7, så skiftas värdena.

0	1	2	3	4
5	3	6	1	7

Därmed var det slut på första skanningen. Nu återstår 4 st. scanningar. Om man studerar adress 3, så inser man att den etta, som ligger där, kommer att ligga längst till vänster, om den vid varje scanning, som återstår flyttar ett steg åt vänster. Den kommer att liksom bubblas till rätt position.

Okey, dags för lite kod. Vi lägger till funktionsprototypen i sortering.h.

sortering.h

```
void bubbelSortering(int*,int antal);
```

Sedan lägger vi till källkoden.

sortering.cpp

```
void bubbelSortering(int* arr,int antal)
{
int temp=0;
    for(int i=0;i<antal;i++)
    {
        for(int j=0;j<antal-1;j++)
        {

                if(arr[j+1]<arr[j])
                {
                cout<<"arr["<<j<<"] och arr["<<(j+1)<<"]skiftar ";
                cout<<"värdena "<<arr[j]<<" och "<<arr[j+1]<<endl;
                temp = arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
```

```
                skrivFormateradArray(arr,antal);
                }


        }
    }
}
```

Nu ska vi förstås göra ett litet program, som använder bubbelsortering.

bubbelsort_program.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include "sortering.h"
using namespace std;

int main()
{
    dos_console();
int data[] ={6,5,3,7,1};
int storlek = sizeof(data)/sizeof(int);
skrivFormateradArray(data,storlek);
bubbelSortering(data,storlek);

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En provkörning av programmet gav följande utskrift:

```
arr[0]->6     ------
arr[1]->5     -----
arr[2]->3     ---
arr[3]->7     -------
```

```
arr[4]->1       -
************************
arr[0] och arr[1]skiftar värdena 6 och 5
arr[0]->5       -----
arr[1]->6       ------
arr[2]->3       ---
arr[3]->7       -------
arr[4]->1       -
************************
arr[1] och arr[2]skiftar värdena 6 och 3
arr[0]->5       -----
arr[1]->3       ---
arr[2]->6       ------
arr[3]->7       -------
arr[4]->1       -
************************
arr[3] och arr[4]skiftar värdena 7 och 1
arr[0]->5       -----
arr[1]->3       ---
arr[2]->6       ------
arr[3]->1       -
arr[4]->7       -------
************************
arr[0] och arr[1]skiftar värdena 5 och 3
arr[0]->3       ---
arr[1]->5       -----
arr[2]->6       ------
arr[3]->1       -
arr[4]->7       -------
```

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

arr[2] och arr[3]skiftar värdena 6 och 1

arr[0]->3       ---

arr[1]->5       -----

arr[2]->1       -

arr[3]->6       ------

arr[4]->7       -------

*************************

arr[1] och arr[2]skiftar värdena 5 och 1

arr[0]->3       ---

arr[1]->1       -

arr[2]->5       -----

arr[3]->6       ------

arr[4]->7       -------

*************************

arr[0] och arr[1]skiftar värdena 3 och 1

arr[0]->1       -

arr[1]->3       ---

arr[2]->5       -----

arr[3]->6       ------

arr[4]->7       -------

*************************

Tryck på en valfri tangent för att fortsätta...
```

Studera skärmutskrifterna. Jag tror det är lättast att följa ettans väg genom arrayen från sista plats till första.

Sortering av strängar

Lika väl som man kan sortera olika typer av tal, så kan man sortera textsträngar. Det gör ingen större skillnad om vi använder Csträngar eller objekt av standardklassen string. Skillnaden är att man kan jämföra två instanser av string med operatoreran >, >=, < och <=, medan man är hänvisad till funktionen strcmp, när det gäller Csträngar. Både Csträngar och instanser av string består av tecken, char. Vi har tidigare konstaterat att char egentligen är en heltalstyp. Det utnyttjas, när man jämför

två texter. Bokstaven A har värdet 65 och har således ett lägre värde än B, dvs. 66. Så för att sortera i bokstavsordning sorterar man i stigande ordning, där ett ord med lägre värde kommer före ett ord med högre.

I exemplet nedan används bubbelsortering. En sorteringsalgoritm, som du redan är bekant med.

sortera_namn_string.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
void printNames(string n[],int a);
void sort(string n[],int a);
int main(int argc, char *argv[])
{
const int ANTAL=4;
string namn[ANTAL]={"Olle","Fredrik","Stefan","Anders"};
    cout<<"Namnen innan sortering"<<endl;
    printNames(namn,ANTAL);
    sort(namn,ANTAL);
    cout<<"Namnen efter sortering"<<endl;
    printNames(namn,ANTAL);
    system("PAUSE");
    return EXIT_SUCCESS;
}

void sort(string n[],int a)
{
string temp;
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<a-1;j++)
        {
            if(n[j]> n[j+1])
```

```
            {
                temp = n[j];
                n[j] = n[j+1];
                n[j+1] = temp;
            }
        }
    }
}
void printNames(string n[],int a)
{
    for(int i=0;i<a;i++)
        cout<<n[i]<<endl;
}
```

Programmet ger följande utskrift:

```
Namnen innan sortering
Olle
Fredrik
Stefan
Anders
Namnen efter sortering
Anders
Fredrik
Olle
Stefan
Tryck på en valfri tangent för att fortsätta...
```

Det hela fungerar ju alldeles utmärkt. Men det ska i ärlighetens namn sägas, att det kan uppstå problem med våra svenska bokstäver Åå, Ää och Öö. Skulle du vara intresserad av att veta mer om problemet, så har jag ägnat hela sista kapitlet, Svenska bokstäver i Windows kommandotolk, åt det.

Jag visar ett exempel med sortering av en array av pekare till char också. Jag har valt att låta funktionen ta argumentet array av pekare till char. Det hade gått precis lika bra att använda pekare till pekare till char (char**). Det beror på att en array aldrig skickas som argument till en funktion.

Det är primäradressen till arrayens första element som skickas. Nu har jag valt att använda en array av pekare till char, för att inte göra dig förvirrad.

I funktionen sortera, så används bubbelsortering. Observera att det är pekare till char vi arbetar med inne funktionen. Det är det som gör att det går att använda ett likamedtecken vid tilldelning av den ena chararrayen till den andra. När man annars tilldelar ett värde till en chararray, så använder man funktionen strcpy(målsträng, källsträng). Men det där sista visste du nog redan.

sortera_namn_cstring.cpp

```
#include <cstdlib>
#include <iostream>
#include <cstdlib>
#include <iostream>using namespace std;
void printNamn(char* n[],int a);
void sortera(char* [],int a);
int main(int argc, char *argv[])
{
    const int ANTAL_NAMN = 4;
    char* namn[ANTAL_NAMN]={"Morgan","Arne","Stefan","Ove"};
    printNamn(namn,ANTAL_NAMN);
    /*Sedan sorterar vi och skriver ut igen.*/
    sortera(namn,ANTAL_NAMN);
    cout<<"Efter sorteringen**************"<<endl;
    printNamn(namn,ANTAL_NAMN);

     system("pause");
     return EXIT_SUCCESS;
}
void sortera(char* namn[],int antal)
{
char *temp;
    for(int i=0;i<antal;i++)
    {
        for(int j=0;j<antal-1;j++)
```

```
        {
            if(strcmp(namn[j],namn[j+1])>0)
            {
            /* Att nedastående fungerar
               beror på att det är pekare vi
               swappar. Annars kan man inte
               tilldela en array till en annan
               med ett likamedtecken =. Då ska
               man istället använda strcpy.*/
                temp = namn[j];
                namn[j] = namn[j+1];
                namn[j+1]=temp;
            }
        }
    }
}

void printNamn(char* n[],int a)
{
    for(int i=0;i<a;i++)
    {
        cout<<n[i]<<endl;
    }
}
```

Programmet ger följande utskrift:

```
Morgan
Arne
Stefan
Ove
Efter sorteringen**********
Arne
```

```
Morgan

Ove

Stefan

Tryck på en valfri tangent för att fortsätta...
```

Det fungerade alldeles utmärkt, eller hur? Men precis, som när vi sorterade strängar med string, så måste jag påpeka att det kan uppstå problem med våra svenska bokstäver Åå, Ää och Öö. Skulle du vara intresserad av att vilja veta mer om problemet, så har jag ägnat hela sista kapitlet, Svenska bokstäver i Windows kommandotolk, åt det.

Strukturer – egendefinierade typer

I C++ och de flesta andra programmeringsspråk har vi olika typer av data. Olika typer passar till olika saker. Om vi skulle göra ett program, som håller reda på diverse mätvärden med decimaldel, så passar double eller float. I andra fall, så kanske ett heltal (int) passar. Men tänk, om man skulle behöva ha datatypen fotboll? Eller datatypen elev. Ja, då finns inga sådana datatyper inbyggda i språket, men man kan faktiskt skapa sådana själv. Då erbjuder C++ en möjlighet att skapa egendefinierade datatyper med en konstruktion, som kallas för struktur. Det är en möjlighet, som C++ har fått med sig från programmeringsspråket C.

Nästa steg i din programmeringsutveckling, som kommer efter den här boken, innebär med all säkerhet, att du ska lära dig programmera objektorienterat. Många moderna språk, C++ inkluderat, är objektorienterade. I Objektorienterad programmering är klass och objekt två viktiga begrepp. Det innebär bland annat, att man kan skapa oerhört avancerade egna datatyper. Det hör till saken, att i C++ anses dessutom struktur och klass vara samma sak. I den här boken, kommer vi längre fram att kika lite på, hur en del av de s.k. standardklasserna kan användas, för att lösa en del programmeringsuppgifter. I övrigt så faller objektorienterad programutveckling utanför ramarna för den här boken.

Låt oss anta, att du i ett program ska hålla reda på och bearbeta uppgifter om rektanglar. En rektangel har ett antal attribut (egenskaper), som kan uttryckas med enkla datatyper. Den har, om man tänker sig att den skulle ritas ut på skärmen, en x och en y position. Dessa uppgfter skulle kunna hanteras med heltal. Vidare, så har en rektangel bredd och höjd. Där skulle man också kunna använda heltal. Baserat på uppgifter om bredd och höjd, så kan man sluta sig till, om rektangeln är kvadratiskt eller inte. Man kan också räkna ut hur lång diagonalen i en rektangel är.

En struktur, som beskriver en rektangel enligt ovan skulle se ut så här:

```
struct Rektangel
{
    int x;
    int y;
```

```
    int width;
    int height;
};
```

Först så inleds det hela med nyckelordet struct åtföljt av strukturens namn. Innanför krullparenteserna ligger strukturens attribut. Lägg märke till att attributen inte har några värden. Rektangeln tjänar som en mall för andra variabler, s.k. kallade strukturobjekt. Observera särskilt, att det ska vara ett semikolon efter den avslutande parentesen.

Nu kan vi skapa strukturobjekt av typen Rektangel. Man kan göra det flera olika sätt. I ett användande program så kan man t.ex. skriva så här:

```
Rektangel rect; //Ett strukturobjekt av typen Rektangel
```

Sedan kan man komma åt strukturens attribut med hjälp av punktoperatorn. Man kan tilldela strukturobjektet attribut värden enligt nedan:

```
    rect.x=50;
    rect.y=50;
    rect.width=100;
    rect.height=100;
```

På samma sätt, så skulle man kunna komma åt strukturobjektens attribut med punktoperatorn, för att skriva ut dess värden.

```
    cout<<"rect.x="<<rect.x<<endl;
    cout<<"rect.y="<<rect.y<<endl;
    cout<<"rect.width="<<rect.width<<endl;
    cout<<"rect.height="<<rect.height<<endl;
```

Man kan som alternativ göra en deklaration och tilldelning på samma gång. Det ser ut så här:

```
Rektangel rect_1={10,10,400,400};
```

Om man har många strukturobjekt t.ex. i en array, så kommer varje strukturobjekt att komma ihåg värdena på sina egna attribut.

Visst kan man ha funktioner, som tar strukturobjekt som argument. Då är det allt som oftast bäst att man utformar funktionen på ett sådant sätt, att funktionen tar en referens till strukturobjekt som argument (Se avsnittet om referenser i denna bok).

Vi kikar väl på ett litet exempel.

rektangel_ett.cpp

```
#include <cstdlib>
#include <iostream>
```

```
#include "rektangel.h"
using namespace std;
struct Rektangel
{
    int x;
    int y;
    int width;
    int height;
};
void skrivRektangel(Rektangel& r);
int main()
{
    Rektangel rect={10,10,400,400};
    skrivRektangel(rect);
    system("PAUSE");
    return EXIT_SUCCESS;
}
void skrivRektangel(Rektangel& r)
{
    cout<<"Rektangelns x="<<r.x<<endl;
    cout<<"Rektangelns x="<<r.y<<endl;
    cout<<"Rektangelns width="<<r.width<<endl;
    cout<<"Rektangelns height="<<r.height<<endl;
}
```

Programmet get följande utskrift:

```
Rektangelns x=10
Rektangelns x=10
Rektangelns width=400
Rektangelns height=400
Tryck på en valfri tangent för att fortsätta...
```

Det finns ju ytterligare information om en rektangel som man kan sluta sig till utifrån den information, som finns om en rektangel. Man kan ta reda på om en rektangel är kvadratisk. Man kan ta reda på det diagonal avståndet. Vidare, så skulle det kunna vara intressant att veta, om en viss punkt befinner sig inom eller utanför en rektangels yta.

Lösa uppgifter med strukturer Rektangel

Vi ska lösa alla dessa uppgifter genom att skapa funktioner, som tar en referens till en rektangel som argument. Det vore lämpligt, om vi la strukturen samt de funktioner, som rör rektanglarna, i en headerfil.

rektangel.h

```
#ifndef _REKTANGEL_H
#define _REKTANGEL_H
#include<iostream>
using namespace std;

/* Först en struktur, som beskriver en rektangel */
struct Rektangel
{
    int x;
    int y;
    int width;
    int height;
};

/*skrivRektangel skriver ut värdena på det
strukturobjekt, som skickas som argument.*/
void skrivRektangel(Rektangel& r);
#endif
```

Nu behöver vi också skriva källkod till funktionen skrivRektangel. Vi lägger koden i en separat källkodsfil.

rektangel.cpp

```
#include "rektangel.h"
void skrivRektangel(Rektangel& r)
```

```
{
    cout<<"Rektangelns x="<<r.x<<endl;
    cout<<"Rektangelns y="<<r.y<<endl;
    cout<<"Rektangelns width="<<r.width<<endl;
    cout<<"Rektangelns height="<<r.height<<endl;
}
```

För att pröva strukturen och funktionen skrivArray, så måste vi skapa ett testprogram.

test_rektangel_one.cpp

```
#include <cstdlib>
#include <iostream>
#include "rektangel.h"
#include<iodos.h>
using namespace std;
int main()
{
dos_console();
Rektangel r = {10,10,100,100};
skrivRektangel(r);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift vid en testkörning:

```
Rektangelns x=10
Rektangelns y=10
Rektangelns width=100
Rektangelns height=100
Tryck på en valfri tangent för att fortsätta...
```

Är rektangeln kvadratisk?

Ja, då har vi lagt en passande grund för vidare äventyr. Vi skulle kunna börja med att undersöka om en rektangel är kvadratisk. En kvadratisk rektangel är en rektangel vars bredd och höjd är lika långa.

Det innebär att vi skulle kunna konstruera en funktion, som tar en referens till rektangel som argument och som returnerar sant om width är lika med height annars falskt. Vi börjar med att komplettera headerfilen.

rektangel.h

```
/*isKvadratisk returnerar sant, om argumentets bredd är
lika med dess höjd, annars falskt.*/
bool isKvadratisk(Rektangel& r);
```

Vi måste naturligtvis skriva källkoden till funktionen i källkodsfilen.

rektangel.cpp

```
bool isKvadratisk(Rektangel& r)
{
    return r.width == r.height;
}
```

Det är nog en ganska bra lösning att uppdatera funktionen skrivRektangel, så att den alltid ger besked om, huruvida rektangeln är kvadratisk. Jag börjar dock med att testa funktionen i mitt testprogram.

test_rektangel_two.cpp

```
#include <cstdlib>
#include <iostream>
#include "rektangel.h"
#include<iodos.h>
using namespace std;
int main()
{
dos_console();
Rektangel r = {10,10,100,100};
skrivRektangel(r);
    if(isKvadratisk(r))
        cout<<"Rektangeln är kvadratisk."<<endl;
    else
        cout<<"Rektangeln är inte kvadratisk"<<endl;
    system("PAUSE");
```

```
    return EXIT_SUCCESS;
}
```

En testkörning av programmet gav följande utskrift:

```
Rektangelns x=10
Rektangelns y=10
Rektangelns width=100
Rektangelns height=100
Rektangeln är kvadratisk.
Tryck på en valfri tangent för att fortsätta...
```

Okey, då bygger vi om funktionen skrivRektangel i källkodsfilen.

rektangel.cpp

```
void skrivRektangel(Rektangel& r)
{
    cout<<"Rektangelns x="<<r.x<<endl;
    cout<<"Rektangelns y="<<r.y<<endl;
    cout<<"Rektangelns width="<<r.width<<endl;
    cout<<"Rektangelns height="<<r.height<<endl;
    if(isKvadratisk(r))
        cout<<"Rektangeln är kvadratisk"<<endl;
    else
        cout<<"Rektangeln är inte kvadratisk."<<endl;
    cout<<"**********************************"<<endl;
}
```

Som du kan se, så passade jag på att lägga dit en utskrift av en linje med stjärnor längst ner i funktionen. Nu kan man rensa bort lite kod från testprogrammet. Vi ska också lägga till ytterligare ett strukturobjekt.

test_rektangel_three.cpp

```
#include <cstdlib>
#include <iostream>
#include "rektangel.h"
#include<iodos.h>
```

```
using namespace std;
int main()
{
dos_console();
Rektangel r = {10,10,100,100};
Rektangel r1 = {50,50,100,75};
skrivRektangel(r);
skrivRektangel(r1);

        system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
Programmet gav följande utskrift:
Rektangelns x=10
Rektangelns y=10
Rektangelns width=100
Rektangelns height=100
Rektangeln är kvadratisk
**********
Rektangelns x=50
Rektangelns y=50
Rektangelns width=100
Rektangelns height=75
Rektangeln är inte kvadratisk.
**********
Tryck på en valfri tangent för att fortsätta...
```

Ligger koordinaterna x och y innanför rektangeln?

Koordinaterna x och y skulle till exempel kunna vara musens koordinater på skärmen. Hur ska man då avgöra om koordinaterna ligger innanför rektangelns yta. Det här är en slags algoritm, som brukar kallas för kollisionskontroll. Skulle du någon gång i framtiden börja programmera spel, så kommer du säkert att skriva den här typen av algoritmer många gånger.

Jo, man skulle kunna börja med att konttrollera x och rektangelns x. Vi kallar hädanefter x för mx och y för my. Om mx är större eller lika med x och mx mindre eller lika x plus rektangelns bredd, så har vi en kollision i x-led. I kod så skulle det kunna se ut så här:

```
ff(mx >= rektangel.x  && mx <= rektangel.x + rektangel.width) {
cout<<”Kollision I x-led”<<endl;}
```

Samma kontroll får man göra även i y-led. Om my är större eller like rektangelns y och my är mindre eller lika med rektangelns y plus rektangelns bredd.

```
If(my>= rektangel.y && my <= rektangel.y + rektangel.height)
{
          cout<<”Kollision i y-led”<<endl;
}
```

För att en kollision ska inträffa fordras att kollision sker både i x-led och i y-led. Vi kan börja med att lägga in funktionsprototypen i headerfilen.

rektangel.h

```
/*inSide returnerar sant, om kordinaterna mx och my
  ligger innanför rektangelns yta */
bool inSide(Rektangel& r,int mx,int my);
```

Sedan skriver vi källkoden I källkodsfilen enligt vårt tidigare resonemang.

rektangel.cpp

```
bool inSide(Rektangel& r,int mx,int my)
{
    return (mx>= r.x && mx <= r.x + r.width &&
                my>=r.y && my <= r.y + r.width);
}
```

Då ska vi förstås testa vår nya funktion i ett program. Jag skapar ett strukturobjekt och slumpar sedan fram ett antal koordinater och testar funktionen på det viset.

test_rektangel_four.cpp

```
#include <cstdlib>
#include <iostream>
#include "rektangel.h"
#include<iodos.h>
#include<ctime>
```

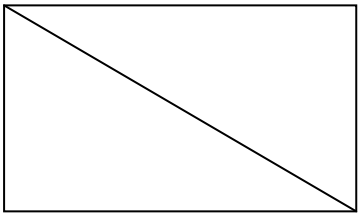
```
using namespace std;
int main()
{
srand(time(0));
dos_console();
int start = 0;
const int MAX=10;
int x=0;
int y=0;
        Rektangel r = {10,10,100,100};
        skrivRektangel(r);
        while(start<MAX)
        {
            x = rand()%200;
            y = rand()%100;
                if(inSide(r,x,y))
                {
                    cout<<x<<" och "<<y;
                    cout<<" kolliderar med rektangeln."<<endl;
                }
                else
                {
                    cout<<x<<" och "<<y;
                    cout<<" ligger utanför rektangeln."<<endl;
                }
            start++;
        }
            system("PAUSE");
    return EXIT_SUCCESS;
}
```

En testkörning av programmet gav följande utskrift:

```
Rektangelns x=10

Rektangelns y=10

Rektangelns width=100

Rektangelns height=100

Rektangeln är kvadratisk

**********

145 och 99 ligger utanför rektangeln.

187 och 83 ligger utanför rektangeln.

180 och 38 ligger utanför rektangeln.

190 och 68 ligger utanför rektangeln.

26 och 6 ligger utanför rektangeln.

92 och 5 ligger utanför rektangeln.

13 och 91 kolliderar med rektangeln.

44 och 93 kolliderar med rektangeln.

174 och 62 ligger utanför rektangeln.

118 och 44 ligger utanför rektangeln.

Tryck på en valfri tangent för att fortsätta...
```

En snabb kontroll av uppgifterna i utskriften visar att vi har kommit rätt i vårt tänk. Det kan inte tillräckligt ofta understrykas, hur viktigt det är att man testar och kontrollerar sina program. Hade vi hittat felaktigheter i utskriften ovan, så hade vi varit tvungna att gå in och felsöka koden efter logiska fel.

Vad är det diagonala avståndet i rektangeln?



Om man tittar noga på figuren ovan, så kan man se, att rektangeln kan delas upp i två lika stora rätsidiga trianglar. Det kan man dra nytta av i det här sammanhanget. Du har kanske lärt dig den gamle matematikern Pythagoras sats, som slår fast, att hypotenusan (diagonalen i bilden ovan) alltid är samma som roten ut summan av sidornas kvadrater. Så för att räkna ut diagonalen, så skulle man kunna tänka på följande sätt: kvadratroten ur sida gånger sida plus sida gånger sida. Eftersom det i

mattebiblioteket finns en funktion, som heter sqrt och som returnerar kvadratroten ur argumentet, så skulle man kunna skriva så här:

```
double diagonal = sqrt( (rektangel.width*rektangel.width)
+(rektangel.height*rektangel.height));
```

Observera att jag har använd datatypen double. Det beror förstås på att det är väldigt sällan, som diagonalen kommer att bli ett jämt heltal.

Vi lägger till en funktion i headerfilen. Vi kan kalla den för getDiagonal.

rektangel.h

```
#include<math.h> //Läggs till längst upp I filen
/* getDiagonal returnerar det diagonal avståndet
   i en rektangel. Returvärdet ges i en double.*/
double getDiagonal(Rektangel& r);
```

Sedan skriver vi källkoden I källkodsfilen. Vi använder den lösning, som jag resonerade mig fram till ovan.

rektangel.cpp

```
double getDiagonal(Rektangel& r)
{
    double retval=0;
    retval = sqrt((r.width*r.width)+(r.height*r.height));
            return retval;
}
```

Jag tror vi lägger in uppgifter om rektangelns diagonal avstånd i funktionen skrivRektangel.

rektangel.cpp

```
void skrivRektangel(Rektangel& r)
{
    cout<<"Rektangelns x="<<r.x<<endl;
    cout<<"Rektangelns y="<<r.y<<endl;
    cout<<"Rektangelns width="<<r.width<<endl;
    cout<<"Rektangelns height="<<r.height<<endl;
/* Nästa rad är ny */
    cout<<"Rektangelns diagonala avstånd = "<<getDiagonal(r)<<endl;
```

```
    if(isKvadratisk(r))
        cout<<"Rektangeln är kvadratisk"<<endl;
    else
        cout<<"Rektangeln är inte kvadratisk."<<endl;
    cout<<"*********************************"<<endl;
}
```

Vi skriver ett litet testprogram även denna gång.

test_rektangel_five.cpp

```
#include <cstdlib>
#include <iostream>
#include "rektangel.h"
#include<iodos.h>
#include<ctime>
using namespace std;
int main()
{
srand(time(0));
dos_console();

        Rektangel r = {10,10,100,100};
        Rektangel r_1 = {100,100,150,100};
        skrivRektangel(r);
        skrivRektangel(r_1);
            system("PAUSE");
    return EXIT_SUCCESS;
}
```

Utskriften vid en testkörning ser ut så här:

```
Rektangelns x=10
Rektangelns y=10
Rektangelns width=100
```

```
Rektangelns height=100

Rektangelns diagonala avstånd = 141.421

Rektangeln är kvadratisk

********************************

Rektangelns x=100

Rektangelns y=100

Rektangelns width=150

Rektangelns height=100

Rektangelns diagonala avstånd = 180.278

Rektangeln är inte kvadratisk.

********************************

Tryck på en valfri tangent för att fortsätta...
```

Några kontroller på min kalkylator visade att vi gjort rätt även denna gång.

elever och provresultat – strukturobjekt

Vi har ju i några tidigare programmeringsexempel i den här boken sysslat med provresultat. Då har provresultaten saknat en viktig ingrediens, nämligen elevernas namn. Vi har enbart bearbetat provresultaten.

Man skulle kunna skapa en struktur, som håller både provresultat och eleven, som presterade resultatet. Då skulle man till exempel använda någon av de sorteringsalgoritmer, som vi studerade tidigare i boken och få ut en sorterad lista över elever och deras provresultat.

Vi börjar med att placera strukturen, som beskriver en elev i en headerfil. En elev har ett namn och ett provresultat. Till namnet skulle vi kunna använda en Csträng och till provresultatet en double.

Vidare, så behöver vi en utskriftsfunktion, som tar en referens till ett strukturobjekt av typen Elev och som skriver ut värdena på objektets attribut. Vi kan kalla funktionen skrivElev.

Vi ska också ha en funktion, som tar en pekare till strukturobjekt av typen Elev. Den ska sortera eleverna efter deras provresultat. Det högsta provresultat ska komma överst. Vi kan kalla funktionen sorteraProv. Vi kan använda bubbelsortering.

elev.h

```
#ifndef _ELEV_H
#define _ELEV_H
#include<iostream>
using namespace std;
```

```
/* Strukturen Elev beskriver en Elev
   med attributen namn och resultat
   (provresultat)*/
struct Elev
{
    char namn[50];
    double resultat;
};
/* skrivElev skriver ut uppgifter
   om en elev. */
void skrivElev(Elev& elev);
/* sorteraProv sorterar elever efter
   deras provresultat. Det bästa
   provresultatet först. */
void sorteraProv(Elev* elever,int antal);
#endif
```

Ja, då skulle vi väl ha lite källkod också. Jag tror inte att det finns några nyheter för din del i den här filen.

elev.cpp

```
#include "elev.h"
using namespace std;
void skrivElev(Elev& elev)
{
    cout<<elev.namn<<"\t:"<<elev.resultat<<endl;
}
void sorteraProv(Elev* elever,int antal)
{
Elev temp;
    for(int i=0;i<antal;i++)
    {
        for(int j=0;j<antal-1;j++)
```

```
        {
            if(elever[j].resultat < elever[j+1].resultat)
            {
                    temp = elever[j];
                    elever[j]=elever[j+1];
                    elever[j+1]=temp;
            }
        }
    }
}
```

Slutligen, så ska vi testa strukturobjektet och våra funktioner I ett program. Jag skapar en array med plats för 5st. elever. Observera att jag använder funktionen strcpy, när jag anger vad eleverna heter. Kom ihåg att man inte kan tilldela chararrayer ett värde med ett likamedtecken. Studera koden noga! Efter programmet så följer en labboration.

provresultat.cpp

```
#include <cstdlib>
#include <iostream>
#include "elev.h"
using namespace std;
int main()
{
    Elev elever[5];
    strcpy(elever[0].namn,"Kalle");
    elever[0].resultat=45;
    strcpy(elever[1].namn,"Olof");
    elever[1].resultat=25;
    strcpy(elever[2].namn,"Arne");
    elever[2].resultat=23;
    strcpy(elever[3].namn,"Kalle");
    elever[3].resultat=11;
    strcpy(elever[4].namn,"Morgan");
    elever[4].resultat=16;
```

```
    int storlek = sizeof(elever)/sizeof(Elev);
    sorteraProv(elever,storlek);
    for(int i=0;i<storlek;i++)
        skrivElev(elever[i]);
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande resultat vid en testkörning.

```
Kalle   :45
Olof    :25
Arne    :23
Morgan  :16
Kalle   :11
Tryck på en valfri tangent för att fortsätta...
```

Usch då! Det där provet gick visst inget vidare (-;

Labboration Elever och testresultat

En labboration är en lite större och mer sammansatt övningsuppgift, som består av ett antal moment.

Du ska förbättra programmet ovan.

1. Lägg till en funktion, som tar en pekare till Elev. Funktionen ska skriva ut samtliga elever med deras namn och resultat. (Det som nu sker i mainfunktionen)
2. Med hjälp av new, så skulle man kunna skapa en dynamisk array med elever. Programmet kan helt enkelt fråga användaren, förmodligen en lärare, hur många testresultat han behöver lägga in i programmet. Du skulle kunna lagra den önskade storleken i en variabel av typen int och kalla den för storlek. Då kan du också ta bort följande rad:

```
int storlek = sizeof(*elever)/sizeof(Elev); // Tas bort
//Storlek blir istället lika med användarens önskemål
```

3. Låt användaren mata in eleverna namn och provresultat. När inmatningen är klar, så presenteras resultatet på skärmen med en sorterad lista.
4. Komplettera med utskrifter av medelvärde och bästa resultat och huruvida elevernas resultat är godkända. Du kan själv bestämma gränsen för godkänt.

Pekare till strukturobjekt

Visst kan man ha pekare till strukturobjekt. Man kan även allokera i det dynamiska minnesutrymmet med new. Den enda skillnaden är, att man inte längre använder punktoperatorn, för att komma åt strukturobjektets attribut, utan piloperatorn ->. Vidare gäller som vanligt att man måste frigöra dynamiskt allokerat minnesutrymme med delete. Jag visar med ett enkelt exempel.

pek_new_strukt.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;
struct Bil
{
string fabrikat;
string regnummer;
int year;
};
void skrivBil(const Bil* b);

int main()
{
    dos_console();
    Bil* b = new Bil;
    b->fabrikat = "Opel";
    b->regnummer = "ADA123";
    b->year = 2004;
    skrivBil(b);
    b->regnummer="CCC123";
    skrivBil(b);
    delete b;

    system("PAUSE");
    return EXIT_SUCCESS;
```

```
}


void skrivBil(const Bil* b)
{
    cout<<"Bilens fabrikat: ";
    cout<<b->fabrikat<<endl;
    cout<<"Bilens registreringsnummer:";
    cout<<b->regnummer<<endl;
    cout<<"Bilens årsmodell: ";
    cout<<b->year<<endl;
    cout<<"********************************"<<endl;
}
```

Observera att jag använt en konstant pekare till Bil, som argument till funktionen skrivBil. Om argumentet är en konstant pekare, så kan man skicka en icke konstant pekare till Bil och en konstant pekare. Hade jag använt en vanlig pekare som argument, så hade jag inte kunnat skicka en konstant pekare. Jag hade då diskvalificerat användningen av konstanta pekare till Bil.

Det vanliga är att skicka strukturobjekt som referenser, men ville jag visa, hur man använder pekare till strukturobjekt. Eftersom funktionen inte ändrar i objektet, så passade det bra med ett konstant pekarargument.

Programmet gav följande utskrift:

```
Bilens fabrikat: Opel
Bilens registreringsnummer:ADA123
Bilens årsmodell: 2004
********************************
Bilens fabrikat: Opel
Bilens registreringsnummer:CCC123
Bilens årsmodell: 2004
********************************
Tryck på en valfri tangent för att fortsätta...
```

Filhantering

Med filhantering menas att skriva till och läsa från fil och i övrigt skapa och ta bort filer. Tyngdpunkten i avsnittet kommer att ligga på att skriva till och läsa från fil.

När jag nu ska introducera dig i filhanteringens konst, så kommer jag att använda någonting, som kallas för klass och som du kan betrakta som avancerade datatyper i stil med strukturer. Vi har redan tidigare i boken sysslat med klassen string. Klass är ett begrepp, som används inom objektorienterad programutveckling, som i och för sig faller utanför ramarna för den här boken. Men vi kommer i ett avsnitt längre fram att syssla med fler klasser från standardbiblioteket. Man kan mycket väl jämföra storheten klass med struktur och i C++, så anses faktisk de båda begreppen beskriva samma sak. Skillnaden mellan klass och struktur består i att klassen, utöver att den har attribut också har funktioner, s.k. medlemsfunktioner.

Filhantering är egentligen ett ganska omfattande område, men jag kommer i det här avsnittet att hålla ett praktiskt betraktelsesätt. Jag kommer helt enkelt att med några praktiska exempel visa, hur man skriver till och läser från fil.

En fil är lika med en mängd bytes data, som hänger samman i en enhet under ett namn. En fil ligger många gånger på ett media för stadigvarande lagring, som en hårddisk, usb osv. En fil kan också ligga i minnet för bearbetning, men vi tänker oss väl för det allra mesta filen, som en enhet som ligger på hårddisken.

Att information ligger i en fil, innebär också att data kan sparas mellan programkörningar. Det är ju förstås någonting, som utnyttjas till alla möjliga typer av problemlösningar. Det som ligger i primärminnet däremot försvinner ju så fort man stänger av datorn.

Klassen för inläsning från fil heter ifstream och klassen för skrivning till fil heter ofstream. Man kommer åt båda dessa klasser genom att inkludera headerfilen fstream. Som ändelsen på klassernas namn antyder, så arbetar dessa med dataströmmar. Dataströmmar har vi redan arbetat med i den här boken, utan att jag för den skull har talat om strömmar. Mellan ditt program och tangentbordet finns det en ström och mellan ditt program och skärmen ligger en annan. Ur operativsystemets synvinkel, så är det ingen större skillnad på en dataström, som är kopplad till tangentbordet och en dataström som är kopplad till en fil för skrivning. Du kommer själv att upptäcka, när du går igenom det här avsnittet, att likheten mellan att skriva till en fil och skärmen är slående. Det gäller naturligtvis även skillnaden mellan att läsa från en fil och tangentbordet. Vi ska börja med ett väldigt enkelt exempel, som visar hur lätt det är att styra en programutskrift från skärmen till en fil.

Att styra utströmmen från skärm till fil

När du har gjort exemplen, som finns i den här boken, så har du förmodligen exekverat programmen i ett IDE. I den här boken har DevC++ använts. Men man kan naturligtvis klicka på den exekverbara filen på hårddisken och få den att exekvera på det viset. Ytterligare ett sätt att köra ett program på, är att starta programmet från kommandotolken eller liknande. I Windows kan man starta kommandotolken genom att välja alternativet [kör] från startmenyn och skriva cmd i det fönster som dyker upp. Då startas kommandotolken. Som regel är den svart och med vit text visas var någonstans bland dina enheter den befinner sig. På min dator, ser det ut så här:

```
Microsoft Windows XP [Version 5.1.2600]

(C) Copyright 1985-2001 Microsoft Corporation


C:\Documents and Settings\morgan.augustsson>
```

Med olika kommandon, som man skriver I kommandotolken och bekräftar med enter, så kan man styra sig mellan mappar och enheter på sin dator. Om jag till exempel skriver CD.., så kommer min kommandotolk att hoppa ur morgan.augustsson och placera sig i Document and Settings. Om jag skriver E: och trycker enter, så hamnar jag på min enhet E:\. På min enhet E:\, så har jag en mapp, som heter test. Genom att skriva CD test, så hamnar jag i mappen test på enheten E:\\. Det ser ut så här:

```
E:\test>
```

Där tänker jag lägga det program som vi strax ska göra.

En sammanställning av användbara kommandon		
CD	Change Directory	Flyttar till angiven mapp
Enhet (c:, E: osv.)		Flyttar till angiven enhet
Dir	Directory	Listar innehållet i en mapp eller på en enhet.
MD (namn)	Make Directory	Skapar en mapp
RMDIR	Remove Direcotry	Ta bort en mapp
HELP		Visar alla kommandon

styrning.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    cout<<"Jag heter Morgan"<<endl;
          return EXIT_SUCCESS;
}
```

Ja, som du kan se, så handlar det om ett jätteenkelt program, som skriver ut ett meddelande på skärmen. Om jag nu spara och kompilerar mitt program i mappen e:\\test, så kan jag köra programmet från kommandotolken.

```
E:\test>styrning

Jag heter Morgan
```

Men jag kan också styra utskriften till en fil. Det kan se ut så här:

```
E:\test>styrning >meddelande.txt
```

Nu skrivs ingenting ut på skärmen. Men om jag tittar i mappen test, så ligger där nu också en textfil, som heter meddelande.txt. I den kan jag läsa, ”Jag heter Morgan”. Det spelade tydligen inte någon som helst roll, att det inte fanns en fil där sedan tidigare. Med tecknet >, så styrdejag utströmmen från programmet till filen meddelande.txt.

Att skriva ett ord till en fil

Vi ska börja med att skriva ett ord till en fil och sedan läsa in filen i ett annat program. Man måste börja med att inkludera <fstream>.

Sedan skapar man ett objekt av typen ofstream (jfr med strukturobjekt), som man kopplar till en fil. Man kan göra det på lite olika sätt.

```
ofstream fout // Kort för file output. Man kan välja vilket namn man vill

fout.open(”fil.txt”); //Öppnar filen för skrivning. Finns inte filen, så skapas en.

/*Alternativt, så kan man öppna filen på en gång */

ofstream fout(”fil.txt”)//Skapar objektet fout och öppnar filen. Finns ingen, så skapas en
```

write_one.cpp

```
#include <cstdlib>

#include <iostream>

#include<fstream>

using namespace std;

int main()

{

    ofstream fout("fil.txt");

    fout<<"Programmering";

    fout.close(); //Behövs egentligen inte, men ser prydligt ut

        system("pause");

    return EXIT_SUCCESS;

}
```

Man kan stänga filen explicit (uttryckligen) med close, dvs. i programmet ovan med fout.close(). Egentligen behövs inte det, eftersom ofstream och ifstream hanterar det själva, när objekten

försvinner, vilken händer t.ex. då programmet avslutas. Men det kan ju finnas tillfällen, när man kan behöva stänga anslutningen till en fil, t.ex. om man behöver objektet, för att öppna en annan fil.

Man måste förstå att close endast stänger filströmmens anslutning till filen. Den tar inte bort ifstream eller ofstreamobjektet.

Om du har kört programmet ovan, så kan du hitta en fil i samma mapp, där du har det exekverbara programmet. Filen innehåller texten ”Programmering”. Man kan också ange, att filen ska skapas någon annanstans. Då får man se upp med de backslash, som Windows använder och eventuellt byta ut dessa mot vanliga slashar. I programmet ovan hade jag kunna skriva: ofstream fout(”e://test/fil.txt”);.

Att läsa in ett ord från en fil

Ja, då ska vi försöka öppna den nyligen skapade filen och läsa in det ord, som står i den, till ett annat program. Det finns en väsentlig skillnad mellan en fil, som man öppnar för skrivning och en, som man öppnar för läsning. Om den fil man försöker öppna för skrivning inte existerar, så reagerar programmet med att skapa filen. Men om man försöker öppna en fil, som inte finns, för läsning, så är det ju ganska meningslöst, om programmet skapar en fil, vilket för övrigt inte heller sker. Det är med all säkerhet så, att det har ganska stor betydelse för programmet, vilken fil som öppnas, eftersom den innehåller data, som är viktiga för programmet. En tom fil är ingen fullgod ersättning. Man bör alltså kontrollera, om den fil man har försökt öppna verkligen existerar innan man går vidare till att läsa in och bearbeta data från den. Man kan använda funktionen good, eller så använder man objektsnamnet. Skulle objektet inte vara anslutet till någon korrekt fil, så håller objektet i sig värdet false.

read_one.cpp

```
#include <cstdlib>
#include <iostream>
#include<fstream>
#include<iodos.h>
using namespace std;
int main()
{
    dos_console();
    ifstream fin("fil.txt");
    char data[100];
        if(fin.good()) //Är filen okey?
        {
            fin>>data;
            //fin.getline(data,100); Fungerar också
```

```
        }
        fin.close();
    cout<<"Filen innehöll ordet: "<<data<<endl;
        system("pause");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift vid en testkörning:

```
Filen innehöll ordet: Programmering

Tryck på en valfri tangent för att fortsätta...
```

Att hämta allt innehåll i en fil på ett enkelt sätt

Det finns ett väldigt enkelt sätt att hämta allt innehåll i en fil på en gång. Objektet av typen ifsteam har tillgång till en funktion, som heter rdbuf, som returnerar en pekare till filbuffern. Med hjälp av den, så kan man komma åt hela filinnehållet på en gång

hela_filen.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<fstream>
using namespace std;
int main()
{
    ifstream fin("hela_filen.cpp");

    if(fin.good())
    {
        cout<<fin.rdbuf();
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En programkörning gav som resultat, att källkoden ovan skrevs ut på skärmen.

Att läsa in innehållet i en fil tecken för tecken

Vi ska öppna en fil, som innehåller lite fler tecken, än vad som behövs för ett ord. Vi ska skriva ut tecknen på skärmen samt räkna hur många tecken filen innehåller. Jag hade helt enkelt tänkt använda filen read_one.cpp. dvs. källkoden till närmast föregående program. När man läser en fil tecken för tecken, så kan man använda funktionen get. Den tar som argument en char, dvs. ett tecken. När funktionen läser in ett tecken, så konsumerar den också tecknet i inströmmen, vilket innebär att nästa gång funktionen anropas, så är det nästa tecken som hämtas, om det finns något mer vill säga. Finns inget mer tecken, så returnerar funktionen false. Vi har då hamnat i läget end of file. Vi ska också använda en funktion i cout, som vi aldrig tidigare har prövat, nämligen put. Funktionen put tar ett tecken, en char, som argument och skriver ut det på skärmen.

read_two.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<fstream>
using namespace std;
int main()
{
    dos_console();
    ifstream fin("read_one.cpp");
    char c;
    int antal_tecken=0;
    if(fin.good())
    {
        while(fin.get(c))
        {
            antal_tecken++;
            cout.put(c);
        }
        cout<<"Filen innehöll "<<antal_tecken<<" tecken."<<endl;
        fin.close();
    }
    system("PAUSE");
    return EXIT_SUCCESS;
```

```
}
```

Programmet gav utöver källkoden till read_one.cpp som utskrift även besked om, hur många tecken filen innehöll.

```
Filen innehöll 433 tecken.

Tryck på en valfri tangent för att fortsätta...
```

Istället för att skriva filens innehåll tecken för tecken till skärmen, så skulle man kunna skriva på en utström kopplad till en annan fil och på det viset få en kopia på ursprungsfilen.

Att skapa en kopia av en fil

För att åstadkomma detta, så behöver vi ha en inström och en utström. Konstigare än så är det inte.

filkopierare_one.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<fstream>
using namespace std;
int main()
{
    dos_console();
    ifstream fin("read_one.cpp");
    ofstream fut("kopia.cpp");
    char c;
    int antal_tecken=0;
    if(fin.good())
    {
        while(fin.get(c))
        {
            antal_tecken++;
            fut.put(c);
        }
        cout<<antal_tecken<<"tecken kopierades."<<endl;
        fin.close();
```

```
        fut.close();
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift:

```
433  tecken kopierades.
Tryck på en valfri tangent för att fortsätta...
```

Men det bästa av allt är att i min arbetsmapp, där programmet körs, så fanns en perfekt kopia av filen read_one.cpp.

Det här programmet skulle väl rent utav kunna vara användbart, bara man på ett vettigare sätt kunde skicka in namnet på den fil, som man vill kopiera, och det namn man vill ge kopian. En lösning skulle kunna vara, att man låter programmet fråga användaren efter dessa namn. Ett annat sätt är att använda radmatningsargument.

Radmatningsargument

Man kan, om man startar program från kommandotolken, skicka med argument till ett program. Det sker på formen: program argument1 argument2 osv. Man får då något, som ser ut som de kommandon man kan skriva i kommandotolken. Man måste då på något sätt kunna komma åt de argument, som användaren ger i kommandotolken. Det kan man genom att utforma mainfunktionen på ett annat sätt:

```
int main(int argc, char *argv[])
{
   system("PAUSE");
    return EXIT_SUCCESS;
}
```

Mainfunktionens parameterlista har utrustats med två argument, ett heltal och en array av pekare till char (eller pekare till pekare till char, om man vill krångla till det). Det första argumentet anger hur många argument, som skickas till programmet och det andra argumentet innehåller de argument som skickas.

Innan jag visar koden till nästa program, så skapar jag en mapp på min enhet E:, som jag döper till argument. Mitt förslag är att du också skapar en särskild mapp till de följande projekten. Skapa mappen på ett ställe, som är lätt att nå från kommandotolken.

argument_one.cpp

```
#include <cstdlib>
```

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    cout<<"argc="<<argc<<endl;
    for(int i=0;i<argc;i++)
        cout<<"argv["<<i<<"]="<<argv[i]<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Om man kör programmet från kommandotolken, så får vi följande utskrift:

```
E:\argument>argument_one
argc=1
argv[0]=argument_one
Tryck på en valfri tangent för att fortsätta...
```

Utskriften betyder att programnamnet, argument_one, räknas som ett programargument. Det innebär att ytterligare argument ligger efter argument 0. Vi prövar:

```
E:\argument>argument_one argument_1 argument_2 argument_3 slutargument
argc=5
argv[0]=argument_one
argv[1]=argument_1
argv[2]=argument_2
argv[3]=argument_3
argv[4]=slutargument
Tryck på en valfri tangent för att fortsätta...
```

När vi nu vet, hur radargument fungerar, så skulle vi kunna bygga om filkopieringsprogrammet, så att man med radmatningsargument kan ange vilken fil, som ska kopieras och vilket namn kopian ska ha.

Filkopiering slutversion

```
#include <cstdlib>
#include <iostream>
```

```
#include<iodos.h>
#include<fstream>
using namespace std;

int main(int argc, char *argv[])
{
dos_console();
    if(argc < 3)
    {
        cout<<"Du måste ange två parametrar ";
        cout<<"[fil att kopiera] [fil att kopiera till]"<<endl;
        system("PAUSE");
            return EXIT_FAILURE;
    }
    ifstream fin(argv[1]);
        if(!fin.good())
        {
            cout<<"Filen du försöker kopiera[";
            cout<<argv[1]<<"] finns inte!"<<endl;
            fin.close();
            system("PAUSE");
             return EXIT_FAILURE;

        }
        else
        {
            ofstream fout(argv[2]);
            char c;
            while(fin.get(c))
            {
                fout.put(c);
```

```
            }
            fout.close();
            fin.close();
        }

    return EXIT_SUCCESS;
}
```

I koden finns två kontroller. Först och främst, så måste minst två argument skickas med, filen som ska kopieras och ett namn på kopian. Det betyder att argc måste ha värdet 3 som lägst. Om så inte skulle vara fallet, så avslutas programmet med en förklaring till användaren.

Den andra kontrollen, som görs, är att filen, som ska kopieras , måste existera. Gör den inte det, vilket kontrolleras med funktionen good, så avslutas programmet med en förklaring till användaren.

Jag gjorde naturligtvis ett par test av programmet, med både felaktigt antal argument och med en fil, som inte fanns som första argument. Slutligen gjorde jag ett test med rätt antal argument och med en fil, som finns, som första argument. Programmet fungerade felfritt.

Att öppna flera filer

Ibland, så kan man behöva öppna fler filer i ett program. Låt oss anta att man skulle vilja räkna sammanlagda antalet tecken i en uppsättning av filer. En variant vore då förstås att öppna filerna med flera ifstremobjekt. Denna modell är förmodligen sämre eftersom den är resurskrävande. Det är bättre att använda samma ifstreamobjekt och öppna, stänga och öppna näst fil osv. Jag hade tänkt utforma programmet på samma sätt som filkopieringsprogrammet, dvs. jag vill att användaren skickar in de filer, vars tecken ska räknas som argument. Man skulle kunna iterera argumenten i en for-loop med start från 1, eftersom det första argumentet är programnamnet. Man kan pröva om argumentet är en giltig fil med funktionen good. Om den returnerar falskt, dvs. filen finns inte, så kan man hoppa till nästa steg i iterationen med continue. Om filen är okey, så läser vi in tecken för tecken och räknar antalet. Innan vi öppnar nästa fil, så stänger vi ifstreamobjektets anslutning till filen med close().

antal_tecken.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<fstream>
using namespace std;
int main(int argc, char *argv[])
{
```

```
dos_console();
int antal_tecken=0;
char c;
ifstream fin;
    for(int i=1;i<argc;i++)
    {
    fin.open(argv[i],ios::in);

        if(!fin.good())
        {
            cout<<"Det gick inte att läsa från " <<argv[i]<<endl;
            fin.close();
            continue;
        }
        cout<<"Läser från "<<argv[i]<<endl;
            while(fin.get(c))
            {
                antal_tecken++;

            }
        fin.close();
    }
    cout<<"Filerna innehöll "<<antal_tecken<<" tecken."<<endl;
    system("pause");
    return EXIT_SUCCESS;
}
```

En provkörning från kommandotolken gav följande utskrift.

```
E:\argument>antal_tecken_ett.cpp main.cpp a.txt
Läser från main.cpp
Det gick inte att läsa från a.txt
Filerna innehöll 703 tecken.
```

```
Tryck på en valfri tangent för att fortsätta...
```

Eftersom jag är säker på att a.txt är namnet på en fil, som ligger i aktuell mapp, så har någonting annat uppenbarligen gått snett. Programmet öppnade första filen, main.cpp, läste igenom den och räknade antal tecken innan den stängdes. Då när programmet stängdes, så befann sig den interna filpekaren i läget eof (end of file). Det innebär, att när good anropades nästa gång, så returnerade den falskt, eftersom strömtillståndet i fin inte var okey, eftersom eof hade uppnåtts. Att man stänger en anslutning till en fil, innebär inte att man tar bort ifstreamobjektet eller på annat sätt återställer ifstreamobjektets tillstånd. För återställa tillståndet i ifstreamobjektet (och ofstreamobjektet också för all del), så kan man anropa funktionen clear(). Jag ändrar i programmet. På raden ovanför filstängningen, så lägger jag till fin.clear().

antal_tecken.cpp

```
while(fin.get(c))
{
     antal_tecken++;
}
  fin.clear(); //Det här anropet lägger du till
  fin.close();
Då testar vi igen:
```

```
E:\argument>argument_one main.cpp a.txt
Läser från main.cpp
Läser från a.txt
Filerna innehöll 754 tecken.
Tryck på en valfri tangent för att fortsätta...
```

Nu fungerade programmet som det ska.

Öppna filer i olika arbetslägen

Hittills, när vi har öppnat filer, så har det varit i läget läsning eller skrivning. Om filen inte har existerar vid skrivning, så har en ny fil skapats. Vidare, så har filpekaren stått i början av filen. Om vi har öppnat en fil skrivning, som har funnits, och denna har haft innehåll, så har den nya skrivningen skrivit över den gamla. Dessa egenskaper har varit filernas arbetsläge. Man kan ställa om arbetsläget, genom att använda ytterligare ett argument, antingen så här ofstream fout(”filnamn”,arbetsläge) eller så här fout.open(”filnamn”,arbetsläge). Här följer en sammanställning över de argument, som kan skickas med. Man kan som regel skicka med flera argument på samma gång. Då gör man det genom att sätta en pipa | mellan argumenten: ofstream(”a.txt”,ios::out||ios::app).

Läge	Betydelse

ios::in	Öppen för läsning
ios::out	Öppen för skrivning
ios::out|Ios::trunc	Öppen för skrivning, skriver över filen, om den finns
ios::out|ios::app	Öppen för skrivning, skrivning hamnar i filslutet
ios::in|ios::out	Öppen för läsning och skrivning
Ios:in|ios::out|ios::trunc	Öppen för läsning och skrivning. Tar först bort filen, om den existerar.
Ios:in|ios::out|ios::app	Öppen för skrivning och läsning. Skrivning sker i slutet av filen.
ios::binary	Filen öppnas i binärt läge.

Att spara uppgifter om e-postadresser i en fil

epost_cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include<fstream>
using namespace std;
int main()
{
    dos_console();
    cout<<"Du matar in valfritt antal poster. X avslutar"<<endl;
    /*Först öppnar vi en fil i lägget app, vilket
      innebär att man skriver i slutet av filen.*/
    ofstream fout("epost.txt",ios::app);
    string namn="";
    string epost;
        while(true)
        {
            cout<<"Namn: ";
            getline(cin,namn,'\n');
                if(namn=="x")
                    break;
            cout<<"Epost :";
```

```
            getline(cin,epost,'\n');
            fout<<namn+":"+epost<<endl;
        }
    fout.close();
    /* Sedan läser vi in innehållet i epostfilen
       och presenterar detta för användaren.*/
    ifstream fin("epost.txt",ios::in);
    if(fin.good())
    {
    cout<<"Epostlista"<<endl;
    cout<<fin.rdbuf()<<endl;
    }
    fin.close();
        system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet börjar med att låta användaren mata in önskat antal e-postadresser. Uttrycket while true innebär egentligen en evighetsloop, men om användare matar in x, så avlutas loopen med break. Alla inmatningar skrivs in längst bak i filen, eftersom vi öppnade den i läget ios::app. Alla utskrifter avslutas med ett radslutstecken, vilket får alla utskrifter till filen att hamna på ny rad. När utskriften till filen avslutats på grund av att användaren har skrivit in ett x, så stängs filen och ett filobjekt för inläsning öppnas. Om den efterfrågade filen finns, så skrivs hela innehållet ut på skärmen med rdbuf, som returnerar en pekare till den buffer, som finns i objektet fin. Slutligen stängs filen.

En programkörning såg ut så här:

```
Du matar in valfritt antal poster. X avslutar

Namn: Lennart Andersson

Epost :lennart.andersson@home.nu

Namn: x

Epostlista

Morgan Augstsson:morgan@hjo.se

Lennart Karlsson:lennart@hjo.se

Per Svensson:per@tibro.se
```

Elizabeth Augustsson:elizabeth@tidahplm.se

Lennart Andersson:lennart.andersson@home.nu

Tryck på en valfri tangent för att fortsätta...

Att öppna en fil för skrivning och en annan för läsning i binärt läge

På de allra flesta operativsystem, så finns det två lägen för filer, text och binärt. Att skriva till och läsa från binärfiler är effektivt, eftersom data inte behöver göras om från datorns naturliga format, det binära, till text. Det finns dock en nackdel, du kan inte läsa binärt. I alla fall inte utan väldigt lång träning (-;

När man skriver till en binärfil, så skrivs helt enkel de bytes, som håller den data du vill spara ner. Detta är väldigt praktiskt, om man skulle vilja spara ner och läsa in hela strukturobjekt till fil. Observera, du kan förvisso spara ner primärminneadresser (pekare), men det är ingen större mening med det, efterson det är föga troligt, att objektet skulle råka få samma primärminnesadress nästa gång programmet körs.

Jag tänkte att vi skulle börja med ett väldigt enkelt program, som skapar ett strukturobjekt av typen person med attributen namn och ålder. Sedan ska vi göra ett program, som läser in samma objekt från filen. Vi börjar med att skapa personstrukturen i en headerfil.

person.h

```
#ifndef _PERSON_H
#define _PERSON_H
struct person
{
    char namn[50];
    int age;
};
#endif
```

Sedan hoppas vi raskt vidare till programfilen. Där skapar vi ett strukturobjekt av typen person. Skriver sedan ut uppgifter om objektets attribut. Sedan öppnar vi en fil i utskrifts och binärläge och skriver ner hela objektet på en gång med funktionen write.

skriv_binary.cpp

```
#include <cstdlib>
#include <iostream>
#include <fstream>
#include<iodos.h>
#include "person.h"
```

```
using namespace std;
int main()
{
    dos_console();
    person p={"Morgan Augustsson",50};
    cout<<"personens namn: "<<p.namn<<endl;
    cout<<"personens ålder: "<<p.age<<endl;

    ofstream fut("person.dat",ios::out|ios::binary);
/* På nästa rad sker skrivningen av hela objektet till filen.
   Man alltid uppnå en sparning till fil av ett objekt på samma
sätt: write((char*)&objekt,sizeof(objekt)) */
    fut.write((char*)&p,sizeof(person));
    fut.close();
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En körning av programmet gav följande resultat:

```
personens namn: Morgan Augustsson
personens ålder: 50
Tryck på en valfri tangent för att fortsätta...
```

Vid en kontroll av min hårddisk, så hittade jag också en fil, som hette person.dat. Nu hoppar vi raskt till nästa program, som läser in strukturobjektet och skriver ut dess attributs värden på skärmen.

person.h

```
[samma som förut]
```

I read_bin öppnar vi en fil i läget in och binary. Sedan läser vi in ett strukturobjekt av typen person med funktionen read.

read_binary.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
```

```
#include<fstream>
#include "person.h"
using namespace std;

int main()
{
    dos_console();
    person p;
    ifstream fin("person.dat",ios::in|ios::binary);
    if(fin.good())
    {
/* På nästa rad sker inläsning av hela objektet från filen.
   Man kan alltid uppnå en inläsning från fil av ett objekt på samma
sätt: read((char*)&objekt,sizeof(objekt)) */

        fin.read((char*)&p,sizeof(person));
        fin.close();
    }
    cout<<"personens namn: "<<p.namn<<endl;
    cout<<"personens ålder: "<<p.age<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En testkörning gav följande utskrift:

```
personens namn: Morgan Augustsson
personens ålder: 50
Tryck på en valfri tangent för att fortsätta...
```

Att öppna en fil för både skrivning och läsning

Vi har tittat på klasserna ifstream för läsning från fil och ofstream för skrivning till fil. Men det finns faktiskt en klass, som möjliggör att man öppnar en fil för både läsning och skrivning. Den heter fstream. Man skulle lite slarvigt uttryckt kunna säga att den är en sammanslagning av ifstream och ofstream. Den har ett par användbara funktioner för förflyttning i en fil, seekg och seekp. Seekg är

till för att placera filpekaren i en position för läsning från filen (g är kort för get) . Seekp används till att placera filpekaren i en position för skrivning (p är kort för print). Båda funktionerna finns i två varianter. Den enklare varianten tar ett heltal, som argument. Det handlar egentligen om heltal av typen streampos, som är en definierad heltalstyp. Heltalet ska vara ett icke negativt tal. Argumentet anger positionen i bytes. Den andra varianten tar två argument av heltalstypen. Det andra argumentet, som kan vara något av följande, ios::beg, ios:: end och ios::cur (början, slutet och nuvarande position), anger en position, som det första argumentet förhåller sig relativt till. Det innebär att man också kan ange negativa tal som första argument. T.ex. seekp(-5,ios::end) innebär 5 bytes från slutet.

Om man vill veta vart man är filen, så kan man använda funktionerna tellg och tellp. Vi tänker oss att vi har en textfil på hårddisken, som heter data.txt och ser ut på följande sätt:

Detta är en textfil, som har

till uppgift, att användas av

fstream i ett exempelprogram

för boken NybörjarProgrammering

i C++.

Vi ska öppna den i programmet nedan och utföra lite operationer på den.

read_and_write.cpp

```
#include <cstdlib>
#include <iostream>
#include<fstream>
#include<iodos.h>
using namespace std;

int main()
{
    dos_console();
   /*ios::ate placerar filpekare längst ner i filen */
   fstream fil("data.txt",ios::in|ios::out|ios::ate);

   if(fil.good())
   {
    cout<<"Skrivläget är på position: "<<fil.tellp()<<endl;
```

```
    cout<<"Läsläget är på position: "<<fil.tellg()<<endl;
    cout<<"Flyttar skrivläget till början på filen."<<endl;
    fil.seekp(0,ios::beg);
    cout<<"Skrivläget är nu på position: "<<fil.tellp()<<endl;
    cout<<"Skriver en rad i början på filen"<<endl;
    fil<<"Skriver denna rad i början på filen."<<endl;
    cout<<"Flyttar läsläget till början på filen."<<endl;
    fil.seekg(0,ios::beg);
    cout<<"Läsläget är nu på position: "<<fil.tellg()<<endl;
    char rad[50];
    fil.getline(rad,50);
    cout<<"Läste in följande rad: "<<rad<<endl;
    cout<<"Flyttar skrivläget till filens slut minus 4 bytes"<<endl;
    fil.seekp(-4,ios::end);
    fil.put('M');
    cout<<"skriver ut hela filens innehåll"<<endl;
    /* Innan så sker, så måste vi flytta filpekare för läsning till
filen start.*/
    fil.seekg(0,ios::beg);
    cout<<fil.rdbuf();
    fil.close();
   }
   system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift:

```
Skrivläget är på position: 130
Läsläget är på position: 130
Flyttar skrivläget till början på filen.
Skrivläget är nu på position: 0
Skriver en rad i början på filen
```

```
Flyttar läsläget till början på filen.

Läslägget är nu på position: 0

Läste in följande rad: Skriver denna rad i början på fi

Flyttar skrivläget till filens slut minus 4 bytes

skriver ut hela filens innehåll

Skriver denna rad i början på filen.

gift, att användas av

fstream i ett exempelprogram

för boken NybörjarProgrammering

i M++.Tryck på en valfri tangent för att fortsätta...
```

Mitt tips till dig är att du studerar programmet och dess utskrift ovan noggrant. Självklart, så ska du själv öppna en fil för läsning och skrivning och laborera med de olika funktionerna, som demonstrerats och förklarats ovan.

Att skapa ett program, som hanterar en databas över personer och deras e-postadresser

Detta exempelprogram är tänkt att bli ett lite större program. Vi vet nu, hur man kan spara ner och läsa in hela strukturobjekt i en binärfil. Vi vet också, hur man ska stega i en fil. Vi skulle alltså kunna göra ett databasprogram, som hanterar uppgifter om personer, deras telefonnummer och e-postadresser. Uppgifter om en person kallas för en post.

Det finns en hel del som man bör kunna göra med personposter, om vi ska kunna kalla det för ett databasprogram. Man ska naturligtvis kunna spara poster, ta bort poster, redigera poster, se poster och söka poster.

Meny

Jag tänkte att vi skulle börja med att skapa en meny, så att användaren själv kan välja, vad han/hon vill göra.

person_databas.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
using namespace std;


int main()
```

```
{
dos_console();
char c='x';
    while(c!='6')
    {
        cout<<"1 Lägg till post."<<endl;
        cout<<"2 Ta bort post."<<endl;
        cout<<"3 Redigera post."<<endl;
        cout<<"4 Sök post."<<endl;
        cout<<"5 Visa alla poster."<<endl;
        cout<<"6 Avsluta."<<endl;
        cin.get(c).get();

        switch(c)
        {
        case '1':
            cout<<"Lägger till post."<<endl;
        break;
        case '2':
            cout<<"Tar bort post."<<endl;
        break;
        case '3':
            cout<<"Redigerar post."<<endl;
        break;
        case '4':
            cout<<"Söker post."<<endl;
        break;
        case '5':
            cout<<"Visar alla poster."<<endl;
        break;
        case '6':
```

```
            cout<<"Avslutar! Hej då"<<endl;
        break;
            default:
            {
            cout<<"Hej då!"<<endl;
            c='6';
            break;
            }

        }

    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

I programmet finns en while-loop, som körs så länge som användarens inmatning i form av en char inte är lika med 6. Studerar man switch-satsen, så kan man faktiskt se, att vilket tecken som helst, som inte är ett giltigt menyalternativ, avbryter while-loopen. Det framgår av etiketten default. Default fångar upp alla värden, som inte har fångats upp av etiketterna ovanför.

När programmet läser in användarens val i menyn, så sker det med funktionen get(char). Funktionen get kan alltså användas, om man behöver läsa in ett enskilt tecken från tangentbordet. Det finns dock ett problem med det, användare skriver inte ett tecken utan två, nämligen valt tecken plus entertecknet. Det innebär att nästa gång man försöker läsa in ett tecken med get(char), så finns redan ett tecken i inströmmen från tangentbordet. Funktionen get(char) returnerar ett objekt av typen istream (cin), vilket gör, att man har tillgång till ett osynligt (namnlöst) objekt, som man kan använda till att anropa andra funktioner, som cin har tillgång till. I programmet, så används det osynliga objektet till att anropa funktionen get. Den funktionen används också till att läsa in ett tecken, (en char). Men till skillnad från get(char), så returnerar funktionen tecknet.

```
char c = cin.get();
```

Så sammanfattningsvis, så innebär nedanstående uttryck helt enkelt, att användarens tangentbordsinmatning lagras i charvariabeln c och att entertecknet konsumeras.

```
cin.get(c).get();
```

Skulle vi enbart skriva, cin.get(c), så skulle användare endast få möjlighet att skriva in sitt menyval varannan gång i while-loopen.

För övrigt, så kan väl sägas att char c tilldelas ett värde, för att while-loopen ska startas. Det kunde ha varit vilket värde som helst utom 6. Till och med om c inte hade fått något värde över huvud taget, så hade det fungerat, men eftersom det är lite osäkert i C++, vad som händer i ett sådant läge, så är det min rekommendation, att du tilldelar c ett värde.

En testkörning av programmet såg ut så här:

```
1 Lägg till post.

2 Ta bort post.

3 Redigera post.

4 Sök post.

5 Visa alla poster.

6 Avsluta.

1

Lägger till post.

1 Lägg till post.

2 Ta bort post.

3 Redigera post.

4 Sök post.

5 Visa alla poster.

6 Avsluta.

6

Avslutar! Hej då

Tryck på en valfri tangent för att fortsätta...
```

Strukturen person och en utskriftsfunktion för personobjekt.

Om man ska skriva ner ett strukturobjekt på fil och för det ändamålet använder filens arbetsläge binary, så skrivs hela strukturobjektet ner, dvs. alla bytes. För att då kunna stega mellan strukturobjektet i filen, så är det av största vikt att alla objekt har samma storlek. Ett sätt att åstadkomma det på är att använda Csträngar. En Csträng är som bekant en chararray med ett avslutande 0-tecken. Om man reserverar t.ex. 50 platser för ett förnamn, så kommer namnet Olle plus 46 tomma tecken att skrivas ner i filen. Om man nu har sådan otur, att man försöker spara ett namn, som har mer än 50 bokstäver, så kommer namnet att kapas efter 50 tecken. Otur! Men å andra sidan, hur många sådana namn finns det egentligen?

Vi utrustar strukturen med en chararray för respektive förnamn, efternamn, telefon och e-post. Vidare så ska vi ha en variabel av heltalstyp i strukturen. Tanken bakom det är att varje

strukturobjekt ska ett id, ett tal som unikt identifierar varje post. Man skulle kunna låta programmet tilldela id:et ett värde.

Funktionen skrivPerson, ska skriva ut värdena på ett strukturobjekts samtliga attribut. Det är lämpligt att vi använder en referens till en person som argument. Eftersom vi inte har förs avsikt att ändra någonting i objektet, så kan vi använda en konstant referens till person. Fördelen vi uppnår med det är att man kan anropa funktionen med både vanliga strukturobjekt såväl som konstanter till personer.

person_1.h (version ett)

```
#ifndef _PERSON_H
#define _PERSON_H
#include<iostream>
using namespace std;
struct person
{
    int id;
    char fnamn[50];
    char enamn[50];
    char telefon[20];
    char epost[50];
};

/*Funktionen skrivPerson skriver ut
  värdena på argumentet attribut. Argumentet
  är en konstant referens till en person.
  Det innebär att man kan anropa funktionen
  med vanliga strukturobjekt såväl som
  konstanter till strukturobjekt som argument.
  En förutsättning för att det ska fungera är
  att man inte utför förändringar på objektet
  i funktionen. */
void skrivPerson(const person& p);
#endif
```

Sedan så ska vi skriva källkoden till skrivPerson.

person.cpp (version 1)

```
#include "person.h"
void skrivPerson(const person& p)
{
    cout<<"ID: "<<p.id<<endl;
    cout<<"Namn: "<<p.fnamn<<" "<<p.enamn<<endl;
    cout<<"Telefon: "<<p.telefon<<endl;
    cout<<"Epost: "<<p.epost<<endl;
}
```

Vi bör väl testa strukturen samt skrivfunktionen i ett program. Inkludera person.h i person_databas.cpp. Längst upp i programmet, så lägger du till följande rader:

```
int id = 1;//Kommer att genereras av en inmatningsfunktion
        person p={id,"Morgan","Auguststsson",
                  "089-236712","morgan@hjo.se"};
        skrivPerson(p);
```

Programmet ska nu ge följande utskrift:

```
Namn: Morgan Auguststsson
Telefon: 089-236712
Epost: morgan@hjo.se
```

Lägga till en person

Det känns väl ganska naturligt med en funktion, som lägger till en post. Vi skapar en funktion, som vi kallar addPerson. Här ska vi på ett säkert sätt läsa in ett personobjekt från tangentbordet. Vi måste också generera ett unikt id för posten. När posten är skapad, så ska den skrivas ner till en fil. Filens namn ska vi lägga in i headerfilen som en konstant till en csträng.

Ett problem, som kommer att uppstå, är att första gången man kör programmet, så finns det ingen fil. Jag har löst det på följande sätt:

1. Jag öppnar filen i läget ios::in. Då kommer inte någon ny fil att skapas, om det inte redan finns en
2. Jag prövar om filen finns med funktionen good. Om den returnerar falskt, så stänger jag filen och öppnar den ånyo i läget ios::out, vilket gör att en ny fil skapas. Jag stänger även denna fil.
3. Om filen inte fanns sedan tidigare, så får variabeln id värdet 1.

4. Om filen fanns sedan tidigare, så öppnar jag ånyo file i läget ios::in. Sedan, så loopar jag igenom filen och undersöker id:et på de poster, som finns i filen. Jag väljer det högsta värdet av dessa och ökar värdet med 1. Jag stänger filen igen.
5. Till sist, så öppnar jag filen i läget ios::out och ios::app. Där låter jag användaren mata in personens förnamn, efternamn, telefon och e-postadress. Slutligen, så skrivs objektet ner i filen. Filen stängs.

Man måste tänka på, att objektet fil inte försvinner bara för att man anropar close. Skulle fil befinna sig i ett felaktigt läge, t.ex. i filslutet (eof), så blir det problem, när man ånyo öppnar en fil. För att återställa objektet fil, så används funktionen clear.

person.h (version 2)

```
#ifndef _PERSON_H
#define _PERSON_H
#include<iostream>
#include<fstream>
using namespace std;
const char filnamn[]="personer.dat";
struct person
{
    int id;
    char fnamn[50];
    char enamn[50];
    char telefon[20];
    char epost[50];
};
void skrivPerson(const person& p);
/* I addPerson lägger användaren in
   uppgifter om en ny person.*/
void addPerson(); //Ny funktion
#endif
```

Observera att på grund av utrymmesskäl, så finns endast kommentarer med till de nya funktionerna. All källkod kommer i en komplett version, när vi är färdiga med programmet.

person.cpp (version 2)

```
#include "person.h"
```

```
void skrivPerson(const person& p)
{
    cout<<"ID: "<<p.id<<endl;
    cout<<"Namn: "<<p.fnamn<<" "<<p.enamn<<endl;
    cout<<"Telefon: "<<p.telefon<<endl;
    cout<<"Epost: "<<p.epost<<endl;
}

void addPerson()
{
int id=0;
bool exists=true;
fstream fil;
    /*Vi börjar med att öppna filen i utskriftsläge
      , för att undersöka om filen finns.
      Skulle den inte finnas, så skapar vi en.
    */
    fil.open(filnamn,ios::in|ios::ate|ios::binary);
    {
        if(!fil.good())
        {
            id = 1;
            exists=false;
            fil.close();
            /* Filen fanns inte, så vi skapar en */
            fil.open(filnamn,ios::out|ios::ate|ios::binary);
            fil.close();
        }
    }
    /* Om filen existerade, så måste vi loopa igenom
       alla poster, för att hitta det id, som har
```

```
      högsta värdet. Sedan ökar vi detta värde med
      ett och får på det viset ett nytt id. */
   if(exists)
   {
       fil.clear();//För att ta bort alla fel i fstreamobjektet
       fil.open(filnamn,ios::in|ios::binary|ios::ate);
       person p;
       fil.clear();
       fil.seekg(0,ios::beg);
       id = 1;
       while(fil.read((char*)&p,sizeof(person)))
       {
          if(p.id>id)
               id = p.id;
       }
       id++;
       fil.close();
   }
   fil.clear();
   fil.open(filnamn,ios::out|ios::binary|ios::app);
   person new_p;
   new_p.id = id;
   cout<<"Förnamn: ";
   cin.getline(new_p.fnamn,50);
   cout<<"Efternamn: ";
   cin.getline(new_p.enamn,50);
   cout<<"Telefon: ";
   cin.getline(new_p.telefon,20);
   cout<<"Epost: ";
   cin.getline(new_p.epost,50);
   fil.write((char*)&new_p,sizeof(person));
```

```
    cout<<"Uppgifterna sparades"<<endl;
    fil.close();
}
```

Sedan, så måste vi förstås lägga in ett funktionsanrop till addPerson i while-loopen i programfilen.

person_databas.cpp

```
        switch(c)
        {
        case '1':
           addPerson(); //Tillägg
        break;
```

En testkörning av programmet gav följande resultat:

```
1 Lägg till post.
2 Ta bort post.
3 Redigera post.
4 Sök post.
5 Visa alla poster.
6 Avsluta.
1
Förnamn: Olle
Efternamn: Persson
Telefon: 074-23478
Epost: ole@borta.nu
Uppgifterna sparades
```

Att visa alla poster

Den här uppgiften borde vara en av de lättare att lösa i det här programmet. Vi gör helt enkelt en funktion, som öppnar aktuell fil i läsläge och går igenom den post för post och skickar varje strukturobjekt till funktionen skrivPerson. Vi lägger alltså till funktionen visaAlla i headerfilen.

person.h

```
/* Funktionen visaAlla öppnar databasfilen
   i läsläge och itererar denna post för post.
```

```
   Varje enskilt strukturobjekt skickas till
   funktionen skrivPerson.*/
void visaAlla(); //Tillägg
```

person.cpp

```
void visaAlla()
{
fstream fil;
    fil.open(filnamn,ios::in|ios::binary);
    if(fil.good())
    {
        fil.seekg(0,ios::beg);
        person p;
        while( fil.read((char*)&p,sizeof(person)))
                skrivPerson(p);
    }
    fil.close();
}
```

Självklart, så måste vi lägga till även det här funktionsanropet på rätt ställe I programfilen.

person_databas.cpp

```
case '5':
            visaAlla();
 break;
```

En testkörning såg ut som nedan på min dator.

```
1 Lägg till post.
2 Ta bort post.
3 Redigera post.
4 Sök post.
5 Visa alla poster.
6 Avsluta.
5
```

```
ID: 1

Namn: Martin Svensson

Telefon: 0504-12387

Epost: martin@home.se

ID: 2

Namn: Olle Persson

Telefon: 074-23478

Epost: ole@borta.nu

ID: 3

Namn: Morgan Augustsson

Telefon: 0503-10714

Epost: morgan.augustsson@live.se
```

Att redigera en post

Här skulle vi kunna använda strategin, att öppna filen i både skriv och läsläge. Sedan itererar vi posterna i filen tills vi hittar posten, som har det id, som användaren har bett att få redigera. Sedan redigerar vid det funna strukturobjektet utifrån de uppgifter, som användaren anger. När det är klart, så flyttar vi utskriftspekaren till platsen för strukturobjektet. Vi kan få tag på punkten vi befinner oss på genom att fråga vart läspunkten ligger, genom tellg. Den punkten måste dock minskas med storleken för ett strukturobjekt av typen person. Sedan skriver vi helt enkelt över det gamla objektet med det redigerade.

Eftersom jag har tänkt lägga in en kontroll av användarens inmatning, när denne anger vilket id, som ska redigeras, så kommer jag att behöver rensa inströmmen från tangentbordet. Jag lägger därför till en bekvämlighetsfunktion, som sköter detta.

Lägg till följande i headerfilen nedanstående i headerfilen.

person.h

```
/*Funktionen clear är en bekvämlighetsfunktion.
  Den rensar inströmmen från tangentbordet.*/
void clear();
/*Funktionen redigeraPerson frågar användaren
  efter vilket id, som ska redigeras.
  Sedan öppnas en fil i utskrift och inläsningsläge.
  När aktuell post hittas, så får användaren redigera
```

```
  uppgiften. Det redigerande objektet skriver över det
  ursprungliga objektet.*/
void redigeraPerson();
```

Så var det återigen dags att uppdatera källkodsfilen.

person.cpp

```
void redigeraPerson()
{
fstream fil;
int id;
/* Börjar med att fråga användaren efter id på den
   post, som ska redigeras. Här sker en kontroll av
   att det är ett giltigt heltal, som matas in.
   Det förhindrar förvisso inte användaren från
   att ange ett id, som inte existerar i databasen, men
   det är en helt annan sak. För att inte få problem
   längre fram i programmet, så anropas funktionen clear, för
   att återställa dataströmmen från tangentbordet.*/
    cout<<"Ange id på den post, du vill redigera!";
    while(!(cin>>id))
    {
        cout<<"Ett godtagbart tal tack"<<endl;
        clear();
    }
    clear();
    /* Sedan öppnas filen, och om den är okey, så läses
       post efter post in, ända tills posterna är slut eller
       begärd post hittas. Då skriv uppgifter om det
       strukturobjektet ut.*/
    fil.open(filnamn,ios::in|ios::out|ios::binary|ios::ate);
    if(fil.good())
    {
```

```
        fil.clear();
        fil.seekg(0,ios::beg);
        person p;
        while( fil.read((char*) &p,sizeof(person)))
        {
            if(p.id == id)
            {
                cout<<"id="<<id<<endl;
                /* Sedan tar vi reda på vart
                   läspekare står i filen.*/
                int plats = fil.tellg();
                /*Platsen för läspekare minskas
                  med storleken för en post.*/
                plats-=sizeof(person);
                /*Uppgifter om begärd post
                  skrivs ut på skärmen.*/
                cout<<"Redigera:"<<endl;
                skrivPerson(p);
                /* Användaren skriver in nya uppgifter*/
                cout<<"\nFörnamn: ";
                cin.getline(p.fnamn,50);
                cout<<"Efternamn: ";
                cin.getline(p.enamn,50);
                cout<<"Telefon: ";
                cin.getline(p.telefon,20);
                cout<<"Epost: ";
                cin.getline(p.epost,50);
                /* För säkerhets skulle återställs
                   eventuella feltillstånd i fstreamobjektet.
                   Skrivpekaren flyttas till platsen för det
                   redigerade objektet och de nya uppgifterna
```

```
                    sparas i filen.*/
                fil.clear();
                fil.seekp(plats);
                fil.write((char*)&p,sizeof(person));
                /*Jovisst, jobbet är klart och vi kan avbryta while-
loopen.*/
                break;
            }
        }
        fil.close();
    }
}
void clear()
{
    cin.clear();
    cin.ignore(100,'\n');
}
```

Slutligen, så läger vi till ett funktionsanrop i programfilen på rätt ställe.

person_databas.cpp

```
case '3':
            redigeraPerson();
break;
```

En testkörning på min dator såg ut så här:

```
1 Lägg till post.
2 Ta bort post.
3 Redigera post.
4 Sök post.
5 Visa alla poster.
6 Avsluta.
3
c=3
```

```
Ange id på den post, du vill redigera!2

id=2

Redigera:

ID: 2

Namn: Allan Larsson

Telefon: 096-123321

Epost: allan@home.nu


Förnamn: Per

Efternamn: Larsson

Telefon: 056-238912

Epost: allan@home.se

1 Lägg till post.

2 Ta bort post.

3 Redigera post.

4 Sök post.

5 Visa alla poster.

6 Avsluta.

5

ID: 1

Namn: Morgan Augustsson

Telefon: 0503-10174

Epost: morgan.augustsson@live.se

ID: 2

Namn: Per Larsson

Telefon: 056-238912

Epost: allan@home.se

ID: 3

Namn: Leopold Augustsson
```

```
Telefon: 0503-10174

Epost: leo@hjo.se

ID: 4

Namn: Kalle Nilsson

Telefon: 045-123789

Epost: kalle@hjo.se
```

Att ta bort en post i databasfilen

När det gäller att ta bort en post ur databasen, så är det inte så lätt att åstadkomma det utan att ta till ett litet småfusk. En idé är att öppna databasfilen i läsläge och en annan, temporär fil, i skrivläge. Sedan läser man in posterna i databasfilen och skriver ner dem i den temporära filen, dvs., alla utom posten, som skulle tas bort. När den operationen är klar och filerna är stängda, kan man ta bort databasfilen och döpa om den temporära filen, så att denna blir databasfil. För att lyckas med detta, så måste vi använda två funktioner, som finns i stdio.h, nämligen remove och rename. Vi börjar med att lägga till funktionen removePerson i headerfilen.

person.h

```
#include<stdio.h> //Lägg till den här inkluderingen
/*Funktionen removePerson tar bort den post i
  databasfilen, vars id har matats in av användare.*/
void removePerson();
```

Så lägger vi i vanlig ordning till källkoden I källkodsfilen.

person.cpp

```
void removePerson()
{
fstream fil;
fstream temp;
int id;
    /* Börjar med att fråga användaren efter id på den
   post, som ska ta bort. Här sker en kontroll av
   att det är ett giltigt heltal, som matas in.
   Det förhindrar förvisso inte användaren från
   att ange ett id, som inte existerar i databasen, men
   det är en helt annan sak. För att inte få problem
```

```
längre fram i programmet, så anropas funktionen clear, för
att återställa dataströmmen från tangentbordet.*/
 cout<<"Ange id på den post, du vill ta bort!";
 while(!(cin>>id))
 {
     cout<<"Ett godtagbart tal tack"<<endl;
     clear();
 }
 clear();
 /* Öppnar databasfilen och flyttar läspekaren
    till filens början.*/
 fil.open(filnamn,ios::in|ios::binary|ios::ate);
 fil.clear();
 fil.seekp(0,ios::beg);
 if(fil.good())
 {
     /*Öppnar en ny fil. Om filen inte finns, så skapas en
       ,eftersom den öppnas i skrivläge. Det behövs ingen
       kontroll av att filen är okey. Vill du göra en sådan
       kontroll, så kan du lägga till det.*/
         temp.open("temp",ios::out|ios::binary|ios::app);

             person p;
        /* Sedan itereras alla poster i databasfilen. Alla
           personobjekt skrivs ner i den temporära filen, utom
           den post, som användare bestämt ska tas bort.*/
             while( fil.read((char*)&p,sizeof(person)))
             {
                 if(p.id != id)
                 {
                     temp.write((char*)&p,sizeof(person));
```

```
                    }
                }

                fil.close();
                temp.close();
                /* Till sist, så tar vi bort databasfilen
                   och döper om den temporära filen till
                   det namn, som databasfilen hade.*/
                remove(filnamn);
                rename("temp",filnamn);

    }
}
```

Vi lägger slutligen till funktionsanropet på rätt ställe i programmet.

person_databas.cpp

```
case '2':
            removePerson();
break;
```

En testkörning på min dator gav följande utskrift:

```
1 Lägg till post.
2 Ta bort post.
3 Redigera post.
4 Sök post.
5 Visa alla poster.
6 Avsluta.
5
ID: 4
Namn: Valfrid Ohlsson
Telefon: 089-34678
Epost: valfrid@fridene.nu
```

```
ID: 5

Namn: Peter Nordström

Telefon: 099-34675

Epost: peter@norr.ny

1 Lägg till post.

2 Ta bort post.

3 Redigera post.

4 Sök post.

5 Visa alla poster.

6 Avsluta.

2

Ange id på den post, du vill ta bort!4

1 Lägg till post.

2 Ta bort post.

3 Redigera post.

4 Sök post.

5 Visa alla poster.

6 Avsluta.

5

ID: 5

Namn: Peter Nordström

Telefon: 099-34675

Epost: peter@norr.ny
```

Att söka efter en post i databasen

Frågan, som vi bör fundera på här är, vilket eller vilka kriteria ska man kunna söka på. Jag kommer att hålla det enkelt, användaren kan söka på efternamn eller förnamn. Det här är säkert någonting, som du skulle kunna utveckla på egen hand. Utöver att man skulle kunna söka på e-postadress och telefon och kombinationer av dessa, så skulle kan också kunna tänka sig sökningar, som är lite bredare, t.ex. hitta alla poster vars förnamn börjar på F.

Vi lägger till en funktionsprototyp i headerfilen.

person.h

```
/*Funktionen search skriver ut de poster, som
  hittas efter en matchning mor användarens
  inmatade sökkritera, Användaren får välja
  i början av funktionen, om det är förnamn
  eller efternamn, som sökningen ska ske efter.
  Det finns även en felhantering, som innebär
  att sökning sker på efternamn, om användaren råkar
  skriva någonting annat än e eller f.
  */
void search();
```

Funktionens källkod lägger vi i vanlig ordning källkodsfilen.

person.cpp

```
void search()
{
    fstream fil;
    person p;
    char c;
    cout<<"Du kan söka på förnamn[f] eller efternamn[e]."<<endl;
    cin.get(c).get();
    if(c!= 'e' && c !='f')
            c='e';
    if(c=='e')
    {
    char enamn[50];
    cout<<"Ange efternamnet du vill söka på!"<<endl;
    cin.getline(enamn,50,'\n');
    fil.open(filnamn,ios::in|ios::binary|ios::ate);
        if(fil.good())
        {
```

```
            fil.clear();
            fil.seekp(0,ios::beg);
            while(fil.read((char*)&p,sizeof(person)))
            {
                if(strcmp(p.enamn,enamn)==0)
                    skrivPerson(p);
            }
        }
    fil.close();

    }
    if(c=='f')
    {
    char fnamn[50];
    cout<<"Ange förnamnet du vill söka på!"<<endl;
    cin.getline(fnamn,50,'\n');
    fil.open(filnamn,ios::in|ios::binary|ios::ate);
        if(fil.good())
        {
            fil.clear();
            fil.seekp(0,ios::beg);
            while(fil.read((char*)&p,sizeof(person)))
            {
                if(strcmp(p.fnamn,fnamn)==0)
                    skrivPerson(p);
            }
        }
    fil.close();
     }
}
```

Slutligen lägger vi ett funktionsanrop I programfilen.

person_databas.cpp

```
case '4':
            search();
break;
```

En testkörning gav följande resultat på min dator:

```
 1 Lägg till post.
2 Ta bort post.
3 Redigera post.
4 Sök post.
5 Visa alla poster.
6 Avsluta.
4
Du kan söka på förnamn[f] eller efternamn[e].
e
Ange efternamnet du vill söka på!
Nordström
ID: 5
Namn: Peter Nordström
Telefon: 099-34675
Epost: peter@norr.ny
ID: 7
Namn: Anders Nordström
Telefon: 078-34789
Epost: anders@tjosan.se
```

Därmed så får väl programmet anses vara färdigt. Det finns säkerligen förbättringar att göra. Det står dig fritt att lägga till finesser i programmet. Slutligen, så en listning över den kompletta koden, personer.h, personer.cpp och person_databas.cpp.

person.h

```
#ifndef _PERSON_H
```

```
#define _PERSON_H
#include<iostream>
#include<fstream>
#include<stdio.h>//Lägg till den här inkluderingen
using namespace std;
const char filnamn[]="personer.dat";

struct person
{
    int id;
    char fnamn[50];
    char enamn[50];
    char telefon[20];
    char epost[50];
};

/*Funktionen redigeraPerson frågar användaren
  efter vilket id, som ska redigeras.
  Sedan öppnas en fil i utskrift och inläsningsläge.
  När aktuell post hittas, så får användaren redigera
  uppgiften. Det redigerande objektet skriver över der
  oredigerade objektet.*/
void redigeraPerson();

/* Funktionen visaAlla öppnar databasfilen
   i läsläge och itererar denna post för post.
   Varje enskilt strukturobjekt skickas till
   funktionen skrivPerson.*/
void visaAlla();

/*Funktionen skrivPerson skriver ut
```

```
  värdena på argumentet attribut. Argumentet
  är en konstant referens till en person.
  Det innebär att man kan anropa funktionen
  med vanliga strukturobjekt såväl som
  konstanter till strukturobjekt som argument.
  En förutsättning för att det ska fungera är
  att man inte utför förändringar på objektet
  i funktionen. */
void skrivPerson(const person& p);

/* I addPerson lägger användaren in
   uppgifter om en ny person.*/
void addPerson();

/*Funktionen removePerson tar bort den post i
  databasfilen, vars id har matats in av användare.*/
void removePerson();

/*Funktionen search skriver det poster, som
  hittas efter en matchning mor användarens
  inmatade sökkritera, Användaren får välj
  i början av funktionen, om det är förnamn
  eller efternamn, som sökningen ska ske efter.
  det finns även en felhantering, som innebär
  att sökning sker på efternamn, om användaren råkar
  skriva någonting annat än e eller f.
  */
void search();

/*Funktionen clear är en bekvämlighetsfunktion.
  Den rensar inströmmen från tangentbordet.*/
```

```
void clear();
#endif
```

person.cpp

```
#include "person.h"
void skrivPerson(const person& p)
{
    cout<<"ID: "<<p.id<<endl;
    cout<<"Namn: "<<p.fnamn<<" "<<p.enamn<<endl;
    cout<<"Telefon: "<<p.telefon<<endl;
    cout<<"Epost: "<<p.epost<<endl;
}
void search()
{
    fstream fil;
    person p;
    char c;
    cout<<"Du kan söka på förnamn[f] eller efternamn[e]."<<endl;
    cin.get(c).get();
    if(c!= 'e' && c !='f')
            c='e';
    if(c=='e')
    {
    char enamn[50];
    cout<<"Ange efternamnet du vill söka på!"<<endl;
    cin.getline(enamn,50,'\n');
    fil.open(filnamn,ios::in|ios::binary|ios::ate);
        if(fil.good())
        {
            fil.clear();
            fil.seekp(0,ios::beg);
            while(fil.read((char*)&p,sizeof(person)))
```

```
            {
                if(strcmp(p.enamn,enamn)==0)
                    skrivPerson(p);
            }
        }
    fil.close();
    }
    if(c=='f')
    {
    char fnamn[50];
    cout<<"Ange förnamnet du vill söka på!"<<endl;
    cin.getline(fnamn,50,'\n');
    fil.open(filnamn,ios::in|ios::binary|ios::ate);
        if(fil.good())
        {
            fil.clear();
            fil.seekp(0,ios::beg);
            while(fil.read((char*)&p,sizeof(person)))
            {
                if(strcmp(p.fnamn,fnamn)==0)
                    skrivPerson(p);
            }
        }
    fil.close();
    }
}

void addPerson()
{
int id=0;
bool exists=true;
```

```
fstream fil;
    /*Vi börjar med att öppna filen i utskriftsläge.
      Det får den effekten att en fil, för att
      undersöka om filen finns. Skulle den inte
      finnas, så skapar vi en.
    */
    fil.open(filnamn,ios::in|ios::ate|ios::binary);
    {
        if(!fil.good())
        {
            id = 1;
            exists=false;
            fil.close();
            /* Filen fanns inte, så vi skapar en */
            fil.open(filnamn,ios::out|ios::ate|ios::binary);
            fil.close();
        }
    }
    /* Om filen existerade, så måste vi loopa igenom
       alla poster, för att hitta det id, som har
       högsta värde. Sedan ökar vi detta värde med
       ett och får på det viset ett nytt id. */
    if(exists)
    {
        fil.clear();//För att ta bort alla fel i fstreamobjektet
        fil.open(filnamn,ios::in|ios::binary|ios::ate);
        person p;
        fil.clear();
        fil.seekg(0,ios::beg);
        id = 1;
        while(fil.read((char*)&p,sizeof(person)))
```

```
        {
           if(p.id>id)
                id = p.id;
        }
        id++;
        fil.close();
    }
    fil.clear();
    fil.open(filnamn,ios::out|ios::binary|ios::app);
    person new_p;
    new_p.id = id;
    cout<<"Förnamn: ";
    cin.getline(new_p.fnamn,50);
    cout<<"Efternamn: ";
    cin.getline(new_p.enamn,50);
    cout<<"Telefon: ";
    cin.getline(new_p.telefon,20);
    cout<<"Epost: ";
    cin.getline(new_p.epost,50);
    fil.write((char*)&new_p,sizeof(person));
    cout<<"Uppgifterna sparades"<<endl;
    fil.close();
}
void redigeraPerson()
{
fstream fil;
int id;
/* Börjar med att fråga användaren efter id på den
   post, som ska redigeras. Här sker en kontroll av
   att det är ett giltigt heltal, som matas in.
   Det förhindrar förvisso inte användaren från
```

```
att ange ett id, som inte existerar i databasen, men
det är en helt annan sak. För att inte få problem
längre fram i programmet, så anropas funktionen clear, för
att återställa dataströmmen från tangentbordet.*/
 cout<<"Ange id på den post, du vill redigera!";
 while(!(cin>>id))
 {
     cout<<"Ett godtagbart tal tack"<<endl;
     clear();
 }
 clear();
 /* Sedan öppnas filen, och om den är okey, så läses
    post efter post in, ända tills posterna är slut eller
    begärd post hittas. Då skriv uppgifter om det
    strukturobjektet ut.*/
 fil.open(filnamn,ios::in|ios::out|ios::binary|ios::ate);
 if(fil.good())
 {
     fil.clear();
     fil.seekg(0,ios::beg);
     person p;
     while( fil.read((char*) &p,sizeof(person)))
     {
         if(p.id == id)
         {
             cout<<"id="<<id<<endl;
             /* Sedan tar vi reda på vart
                läspekare står i filen.*/
             int plats = fil.tellg();
             /*Platsen för läspekare minskas
               med storleken för en post.*/
```

```
                plats-=sizeof(person);
                /*Uppgifter om begärd post
                  skrivs ut på skärmen.*/
                cout<<"Redigera:"<<endl;
                skrivPerson(p);
                /* Användaren skriver in nya uppgifter*/
                cout<<"\nFörnamn: ";
                cin.getline(p.fnamn,50);
                cout<<"Efternamn: ";
                cin.getline(p.enamn,50);
                cout<<"Telefon: ";
                cin.getline(p.telefon,20);
                cout<<"Epost: ";
                cin.getline(p.epost,50);
                /* För säkerhets skulle återställs
                   eventuella feltillstånd i fstreamobjektet.
                   Skrivpekaren flyttas till platsen för det
                   redigerade objektet och de nya uppgifterna
                   sparas i filen.*/
                fil.clear();
                fil.seekp(plats);
                fil.write((char*)&p,sizeof(person));
                /*Jovisst, jobbet är klar och vi kan avbryta while-
loopen.*/
                break;
            }
        }
        fil.close();
    }
}
void visaAlla()
{
```

```
fstream fil;
    fil.open(filnamn,ios::in|ios::binary);
    if(fil.good())
    {
        fil.seekg(0,ios::beg);
        person p;
        while( fil.read((char*)&p,sizeof(person)))
                skrivPerson(p);
    }
    fil.close();
}
void removePerson()
{
fstream fil;
fstream temp;
int id;
    /* Börjar med att fråga användaren efter id på den
   post, som ska ta bort. Här sker en kontroll av
   att det är ett giltigt heltal, som matas in.
   Det förhindrar förvisso inte användaren från
   att ange ett id, som inte existerar i databasen, men
   det är en helt annan sak. För att inte få problem
   längre fram i programmet, så anropas funktionen clear, för
   att återställa dataströmmen från tangentbordet.*/
    cout<<"Ange id på den post, du vill ta bort!";
    while(!(cin>>id))
    {
        cout<<"Ett godtagbart tal tack"<<endl;
        clear();
    }
    clear();
```

```
/* Öppnar databasfilen och flyttar läspekaren
   till filens början.*/
fil.open(filnamn,ios::in|ios::binary|ios::ate);
fil.clear();
fil.seekp(0,ios::beg);
if(fil.good())
{
    /*Öppnar en ny fil. Om filen inte finns, så skapas en
      ,eftersom den öppnas i skrivläge. Det behövs ingen
      kontroll av, att filen är okey. Vill du göra en sådan
      kontroll, så kan du lägga till det.*/
        temp.open("temp",ios::out|ios::binary|ios::app);

            person p;
       /* Sedan itereras alla poster i databasfilen. Alla
          personobjekt skrivs ner i den temporära filen, utom
          den post, som användaren bestämt ska tas bort.*/
            while( fil.read((char*)&p,sizeof(person)))
            {
                if(p.id != id)
                {
                    temp.write((char*)&p,sizeof(person));
                }
            }

            fil.close();
            temp.close();
            /* Till sist, så tar vi bort databasfilen
               och döper om den temporära filen till
               det namn, som databasfilen hade.*/
            remove(filnamn);
```

```
                rename("temp",filnamn);
    }
}
void clear()
{
    cin.clear();
    cin.ignore(100,'\n');
}
```

person_databas.cpp

```
#include <cstdlib>
#include <iostream>
#include<iodos.h>
#include "person.h"
using namespace std;

int main()
{
dos_console();
char c='x';

    while(c!='6')
    {
        cout<<"1 Lägg till post."<<endl;
        cout<<"2 Ta bort post."<<endl;
        cout<<"3 Redigera post."<<endl;
        cout<<"4 Sök post."<<endl;
        cout<<"5 Visa alla poster."<<endl;
        cout<<"6 Avsluta."<<endl;
        cin.get(c).get();

        switch(c)
```

```
        {
        case '1':
           addPerson();
        break;
        case '2':
            removePerson();
        break;
        case '3':
            redigeraPerson();
         break;
        case '4':
            search();
        break;
        case '5':
            visaAlla();
        break;
        case '6':
            cout<<"Avslutar! Hej då"<<endl;
        break;
            default:
            {
            cout<<"Hej då!"<<endl;
            c='6';
            break;
            }

        }

    }
    system("PAUSE");
    return EXIT_SUCCESS;
```

```
}
```

Dynamiska arrayer

Vi har tidigare i boken använt någonting, som jag har kallat för standardklasser. Nu, när du har förståelse för strukturer, så kan sägas att klasser fungerar som väldigt avancerade strukturer. Utöver att de har attribut, precis som strukturer, så har de också funktioner.

Vi har använd ett flertal klasser i den här boken, string, ifstream, ifstream, fstream, istream och ostream. De två objekten cout och cin är av klasserna ostream respektive istream. Man säger att ett objekt är en instans av en klass. Det är att jämföra med att ett strukturobjekt är av viss typ, eller struktur.

Objektorienterad programutveckling faller utanför ramarna för den här boken, men det är nästan omöjligt att undervisa och skriva kod i C++, utan att använda standardklasserna. Att använda standardklasser, skulle kunna ses som ett första steg i din programmeringsutveckling mot objektorienterad programmering. För självklart är det så, att när du behärskar innehållet i den här boken, så är nästa steg att studera objektorienterad programmering i C++.

Vi har tidigare dryftat problematiken med storleken på arrayer, om det är så att man inte på förhand kan säga, hur många platser, som kommer att behövas i en array. Tidigare löste vi det här problemet genom att allokera minne med new i det dynamiska minnesutrymmet. Standardklasserna i C++ erbjuder andra eleganta lösningar. Vi börjar med klassen vector.

Standardklassen vector

För att komma åt klassen vector, så måste man inkludera headerfilen vector. Sedan börjar med att skapa en instans av en vektor. Det gör man på ett sätt, som starkt påminner om deklarationen av en variabel, eller varför inte ett strukturobjekt av viss typ.

Det finn dock en detalj, som är lite speciell, när man skapar en instans av vector, man måste tala om, vad man har tänkt sig att lagra i den. Där skiljer sig inte en vektor åt från en vanligt array, men man gör det på ett alldeles speciellt sätt:

```
vector<int> tal;
```

Alldeles efter klassnamnet, så skriver man < den datatyp, som man har tänkt sig att lagra I vektorn. Man avslutar med >.

Man kan ställa sig frågan, vad är nu detta för konstig syntax?

Jo, i C++ har man någonting, som man kallar för generiska klasser. Vi såg i ett tidigare avsnitt i boken, att man kunde skapa generiska funktioner (mallfunktioner). Generisk betyder generell, dvs. allmän. Eftersom man som programmerare kan tänkas behöva lagra allt från heltal till fotbollar i en vektor, så erbjuder man med hjälp av generella klasser en möjlighet att göra just detta. En annan svensk beskrivning av uttrycket skulle kunna vara mallklasser. (Visst är de både mallar och lite malliga för det!)

När man använder en generisk klass, så anger man inom ett mindre än och ett större än tecken, vad är för typ av data som ska lagras. Vad, som händer, när ett program som använder t.ex. vector<int> kompileras, är att kompilatorn skapar en vektorklass, som hanterar heltal (int). Då var det väl dags att kika på ett exempel.

vector_1.cpp

```
#include <cstdlib>
#include <iostream>
#include<vector> //Obs nödvändigt
using namespace std;
int main()
{
/*Först, så skapas en instans av en vector,
  som ska innehålla heltal (int). Instansens namn är tal.*/
vector<int> tal;
srand(time(0)); //Här genereras en ny slumptalsintervall.
/* På raden nedanför, så slumpas ett
   tal fram. Talet lagras i variabeln tak. Värdet
   på tak anger, hur många heltal, som ska lagras i
   i vektorn. Observera att detta, om inget konstigt
   inträffar, kommer att variera från programkörning
   till programkörning.*/
int tak = (rand()% 150)+100;
int start = 0;
    while(start < tak)
    {
        /*vector har en funktion, som heter
          push_back. Den får argumentet att
          hamna längst bak i vektorn. Här nedan
          slumpas ett tal fram, som ska läggas in.*/
        tal.push_back((rand()%10)+1);
        start++;
    }
```

```
    /* Vektorns funktion size talar om, hur många element
       det finns i vektorn.*/
    cout<<"Det finns "<<tal.size()<<" heltal i vektorn."<<endl;
    /*Här nedan loopas vektorn igenom med hjälp av en
      for-loop. Observera att vanlig arrayindexering kan
      användas på en vektor. Alternativ finns, som vi ska se
      längre fram.*/
    for(int i=0;i<tal.size();i++)
        cout<<"tal["<<i<<"]="<<tal[i]<<endl;


    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Nedan kan du se resultatet av testkörning av programmet på min dator. Allt finns inte med.

```
Det finns 172 heltal i vektorn.
tal[0]=2
tal[1]=8
tal[2]=8
tal[3]=1
tal[4]=8
osv.
```

Att vända på ett ord med hjälp av deque

En annan användbar klass, som påminner mycket om vector är att deque Den fungerar på precis samma sätt som vector, men har funktionen push_front, som vector inte har.

Ett sätt att lägga in någonting i en deque är, som sagt, att använda funktionen push_front. Som namnet antyder, så hamnar det inlagda elementet längst fram i en deque. Om man skulle använda en sträng, t.ex. klassen string och tilldela den värdet Morgan Augustsson och lägga in tecken för tecken i en deque med push_front, så måste resultatet bli, att bokstäverna kommer i omvänd ordning. Itererar man sedan en deque åt rätt håll och skriver ut bokstäverna på skärmen, så får man mitt tjusiga namn (ja, men vadå?) baklänges. Självklart, så kan jag inte hindra, att du använder någon annan text, t.ex. ditt tjusiga namn.

deque_reverse.cpp

```
#include <cstdlib>
#include <iostream>
#include<deque> //Obs nödvändigt
using namespace std;
int main()
{
deque<char> tecken;
string namn="Morgan Augustsson";
    for(int i=0;i<namn.size();i++)
          tecken.push_front(namn[i]);

    for(int i=0;i<tecken.size();i++)
        cout<<tecken[i];
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
nosstsuguA nagroM
Tryck på en valfri tangent för att fortsätta...
```

Att vända på ett ord med hjälp av en stack

En stack fungerar ungefär som en trave tallrikar i disken. Middagsgästerna ställer sina använda tallrikar överst i disktraven. Det sker med funktionen push. Diskaren tar, så länge som det finns tallrikar kvar att diska, översta tallriken. Det sker med funktionerna top och pop.

```
stack_reverse.cpp
#include <cstdlib>
#include <iostream>
#include<stack> //Obs nödvändigt
using namespace std;


int main()
{
```

```
stack<char> tecken;
string namn="Morgan Augustsson";
    for(int i=0;i<namn.size();i++)
          tecken.push(namn[i]); //Lägger in tecken först i stacken
    /* While-loopen här nedan itererar, så länge som det finns
       Någonting kvar i stacken. Pop tar bort det översta
       Element. Det innebär att innehållet tar slut så småningom.*/
    while(!tecken.empty())
    {
      cout<<tecken.top();//Returnerar det översta elementet
      tecken.pop();//Tar bort det översta elementet
    }
     cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
nosstsuguA nagroM

Tryck på en valfri tangent för att fortsätta...
```

Att iterera en vektor med hjälp av en iterator

Du har tidigare i boken lärt dig att pekare och arrayer är intimt förbundna. När det gäller standardklasserna, vector, deque och stack, så finns det en slags pekare, som kallas för iteratorer. Det är en ganska avancerad konstruktion och området är ganska omfattande, så jag tänker inte gå så väldigt djupt in på ämnet, men jag tänker visa hur man kan använda iteratorer. Det finns en hel del programmeringsuppgifter, som man kan få gratis, när man använder standardklasserna, men de förutsätter, att man vet lite om hur man använder iteratorer.

Man får tag på en iterator, via objektens funktioner.

```
vector<int> tal;
vector<int>::iterator it = tal.begin();
```

Vi får alltså tag på den iterator, som finns I objektet tal, genom att anropa funktionen begin. Med hjälp av en iterator, så kan man loopa igenom vektorns element genom att räkna upp den med ++.

Eftersom vektorn ovan pekar på vektorns första element, så kommer it++ innebära att iteratorn pekar på vektorns andra element, om det nu finns ett sådant. Vi ska se på ett litet exempel, som visar hur man itererar en vektor med hjälp av en iterator.

vektor_iterator.cpp

```
#include <cstdlib>
#include <iostream>
#include<vector>
using namespace std;
int main()
{
    /* Programmet börjar med att en vektor
       skapas och en iterator deklareras.
       Sedan fylls vektorn med lite tal.
    */
    vector<int> tal;
    vector<int>::iterator it;
    int i=0;
    while(i++<10)
        tal.push_back(i*5);

    /* For-loopen börjar med att iteratorn
       initieras med hjälp av begin. Så länge
       som iteratorn inte är lika med end(), så sker
       en förflyttning i datamängden med it++*/
    for(it = tal.begin();it != tal.end();it++)
            cout<<*it<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Att funktionen begin ger en iterator, som pekar på samlingens första element är kanske inte så konstigt, men att end inte pekar på datasamlingens sista element, utan på ett tänkt element, som

ligger strax utanför samlingen är kanske lite konstigt. Tanken bakom den designen är att man ska kunna utföra den här typen av iterationer, som vi sett ett exempel på i programmet ovan.

Observera vidare, att när vi ska skriva ut innehållet i vektorn, så skriver vi cout<<*it. VI skriver alltså ut det värde, som iteratorn pekar på.

Ytterligare en standardklass, list

En list (lista) fungerar ungefär som en deque eller vector. Inuti, så är den dock lite annorlunda konstruerad. Den arbetar med en länkad lista. Detta gör listan synnerligen lämplig för vissa operationer. En sådan operation är att man ganska lätt kan ta bort element ur datasamlingen. Vi ska kika på ett exempel där vi tar bort ganska många element ur en lista. Vidare kan nämnas, att om man använder en vector eller deque, så kan man använda arrayindexering, när man arbetar med de enskilda elementen i dessa. Den möjligheten finns inte i list, utan man är hänvisad till att använda en iterator.

lista_tabort_namn.cpp

```
#include <cstdlib>
#include <iostream>
#include<list>
using namespace std;
int main()
{
    list<string> namn;
    list<string>::iterator it;
    string fnamn[]={"Morgan","Anders","Peter","Stefan","Anna"};
    /*Namnen i arrayen med strängarna ovan, fnamn itereras och
      namnen läggs in i listan.*/
    for(int i=0;i<5;i++)
        namn.push_back(fnamn[i]);

    /* Listan itereras med en iterator och namnen
       skrivs ut.*/
    for(it = namn.begin();it != namn.end();it++)
            cout<<*it<<endl;
```

```
     /*Här nedan itereras listans element, dvs. namnen.
       Namn, som börjar på A, tas bort. För att det inte
       ska bli hål i listan under iterationen i for-loopen
       , så måste man använda konstruktionen it =
       list.erase(iterator).
       Missar man detta, så kan for-loopen inte fortsätta och man
       får som regel en programkrasch.*/
    for(it = namn.begin();it != namn.end(); it++)
    {
        string n = *it;
        if(n[0]=='A')
        {
           it =  namn.erase(it);//Läs ovan för mer info.
        }
    }
    /*Slutligen så skrivs listan element ut igen. Detta
      för att kontrollera att inga namn, som börjar på A
      finns kvar.*/
    cout<<"*********************************"<<endl;
    for(it = namn.begin();it != namn.end();it++)
            cout<<*it<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
Morgan
Anders
Peter
Stefan
Anna
*********************************
```

```
Morgan
Peter
Stefan
Tryck på en valfri tangent för att fortsätta...
```

Att sortera namn med list

list har tillgång till en väldigt bekväm funktion, som sorterar dess innehåll. Sorteringen sker i en någorlunda naturlig ordning. Det innebär att bokstäver sorteras i bokstavordning. Hur väl detta fungerar med våra svenska bokstäver, å, ä och ö är kanske en aning beroende på vilken dator man kör programmet på. Jag gjorde ett test på min dator och använde iodos. Det verkade fungera, men bli inte alltför besviken, om det inte fungerar på din dator. Jag har för övrigt ägnat hela sista kapitlet, Svenska bokstäver i Windows kommandotolk, åt problem, som kan uppstå med våra svenska tecken i kommandotolken.

lista_sortera_namn.cpp

```
#include <cstdlib>
#include <iostream>
#include<list>
#include<iodos.h>
using namespace std;

int main()
{
    dos_console();
    list<string> namn;
    list<string>::iterator it;
    string fnamn[]={"Morgan","Anders","Åke","Stefan","Örjan",
                    "Per","Bjarne"};
    /*Namnen i arrayen med strängarna ovan, fnamn itereras och
      namnen läggs in i listan.*/
    for(int i=0;i<7;i++)
        namn.push_back(fnamn[i]);

    /* Listan itereras med en iterator och namnen
       skrivs ut.*/
```

```
    for(it = namn.begin();it != namn.end();it++)
            cout<<*it<<endl;
     namn.sort();
    /*Slutligen så skrivs listan element ut igen. Detta
      för att kontrollera att namnen ligger i alfabetisk ordning.*/
    cout<<"*******Efter sorteringen*********"<<endl;
    for(it = namn.begin();it != namn.end();it++)
            cout<<*it<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav följande utskrift på mind dator:

```
Morgan
Anders
Åke
Stefan
Örjan
Per
Bjarne
********************************
Anders
Bjarne
Morgan
Per
Stefan
Åke
Örjan
Tryck på en valfri tangent för att fortsätta...
```

Att sortera heltal med listfunktionen sort, samt vända på data.

Om man sorterar heltal enligt den naturliga principen, så sorteras siffrorna stigande. Om man skulle vilja ha siffrorna åt andra hållet, så skulle man kunna börja med ett anrop till sort och följa upp med anrop till reverse.

number_sort.cpp

```
#include <cstdlib>
#include <iostream>
#include<list>
using namespace std;
int main()
{
list<int> siffror;
const int ANTAL=10;
int start=0;
    srand(time(0));
    /*Först fylls listan med 10 slumpmässiga värden*/
    while(start++ < ANTAL)
        siffror.push_back(rand()%20);
    /*Här nedan skrivs listans innehåll. Observera att deklarationen
      och tilldelning av iteratorn sker i for-loopen direkt.*/
 for(list<int>::iterator it= siffror.begin();it != siffror.end();it++)
            cout<<*it<<endl;
    /* Sedan sorteras listan och skrivs ut.*/
    siffror.sort();
    cout<<"**************Efter sortering*****************"<<endl;
   for(list<int>::iterator it= siffror.begin();it != siffror.end();it++)
            cout<<*it<<endl;
    /* Slutligen vänds listan åt andra hållet. En ny utskrift av
       listans innehåll följer.*/
    siffror.reverse();
    cout<<"**************Efter vändningen*****************"<<endl;
   for(list<int>::iterator it= siffror.begin();it != siffror.end();it++)
```

```
            cout<<*it<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

En körning av programmet på min dator gav följande utskrift:

```
7
6
17
13
17
3
11
3
10
10
**********Efter sortering**********
3
3
6
7
10
10
11
13
17
17
**********Efter vändningen**********
17
17
```

```
13

11

10

10

7

6

3

3

Tryck på en valfri tangent för att fortsätta...

Formatering av
```

Att ta bort dubbletter i en lista med unique

Funktionen unique går igenom listan och tar bort dubbletter. Vi kan pröva funktionen genom att lägga till den sist i programmet ovan.

```
siffror.reverse();

siffror.unique(); //Lägg till det här funktionsanropet.

    cout<<"**************Efter vändningen*****************"<<endl;

    for(list<int>::iterator it= siffror.begin();it != 
siffror.end();it++)

            cout<<*it<<endl;
```

Här återger slutet på en testkörning av programmet.

```
**************Efter sortering*****************

0

2

3

5

7

8

9

11

16
```

```
16
**********Efter vändningen**********
16
11
9
8
7
5
3
2
0
```

Vi kan se att siffran 16 förekom två gånger i den första utskriften. Anropet till unique tog bort den ena av dessa.

Formaterad utmatning

Vi har genom i stort sett hela boken använt oss av cout. cout är en instans, eller objekt, av standardklassen ostream. Nu ska vi kika på några olika sätt att formatera utdata. Det finns flera olika sätt och ämnet i sig är ganska vittomfattande och jag har inte tänkt gå igenom alla utskriftsmanipulatorer och flaggor. Jag har istället tänkt ge några handfasta exempel på, hur man kan formatera utskrifter . Ett exempel på detta har vi sett tidigare i boken, nämligen hur man kan bestämma antalet decimaler som skrivs ut i ett decimaltal. Vi ska börja med det.

decimaler.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    double a=1;
    double b=3;
    double tal = a/b;
    cout<<"tal="<<tal<<endl;
```

```
    cout<<"tal="<<setprecision(2)<<tal<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Om man beräknar värdet på tal (1/3(på en kalkylator, så får man följande värde: 0,33333333333333333333333333333333. Man får oändligt många decimaler. I programmet ovan skriv tal först ut, utan att något annat format än defaultformatet används. Det innebär att tal skrivs ut med 6 tecken förutom punkten. I andra utskriften används funktionen setprecision med argumentet 2. Det innebär att tal skrivs ut med 2 decimaler. Observera vidare att programmet inkluderar iomanip.

Programmet gav följande utskrift:

```
tal=0.333333

tal=0.33

Tryck på en valfri tangent för att fortsätta...
```

Om man skulle skriva ut diverse priser på varor i ett program, så vill man förstås, att det ska se prydligt ut. Titta på nedanstående program:

priser_1.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    double priser[]={0.50,15.50,13};
    for(int i=0;i<sizeof(priser)/sizeof(double);i++)
            cout<<"Pris: "<<priser[i]<<":kr"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
Pris: 0.5:kr

Pris: 15.5:kr
```

```
Pris: 13:kr

Tryck på en valfri tangent för att fortsätta...
```

Detta är nog inte den utskrift av priser, som vi helst skulle vilja ha. Vi ska därför försöka formatera utskriften. För det första, så skulle det vara snyggt om alla decimaler skrevs ut även nollor, som i 13.50:kr.

priser_2cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    double priser[]={0.50,15.50,13};
    for(int i=0;i<sizeof(priser)/sizeof(double);i++)
            cout<<"Pris: "<<setprecision(2)<<priser[i]<<":kr"<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger nu följande utskrift:

```
Pris: 0.5:kr

Pris: 16:kr

Pris: 13:kr

Tryck på en valfri tangent för att fortsätta...
```

Inte riktigt vad vi hade tänkt oss. Men om vi tittar noga, så kan vi se, att alla prisangivelser innehåller exakt 2 tal. Det betyder att setprecision(2) inte har fungerat riktigt som vi tänkt oss. Det är en helt riktig slutsats. Setprecision fungerar olika i olika lägen. Ibland betyder den, att två decimaler ska användas, ibland betyder den att två tal ska användas totalt. Det finns totalt tre utskriftslägen, fixed, scientific och default. Scientific anger, att utskrift av decimaltal ska ske på den flytande formen 999.99e99. Det här en form, som vi inte har gått igenom i boken, men bokstaven e betyder upphöjt till. Fixed betyder att utskrift sker på fast form med decimalpunkt. Den tredje, defaultläget, det läge, som vi för det mesta har använt i den här boken, innebär att decimaltal skrivs ut med sex tal totalt. Det såg vi ett exempel på i decimaler.cpp. Om man använder funktionen setprecision och utskriften befinner sig i defaultläget, så anger argumentet, som man skickar med i funktionsanropet, hur många tal, som ska användas totalt sett, om det går. I programmet ovan skickade vi med 2. Då gör

programmet helt enkelt om 15.50 till 16, eftersom det består av två tal, 1 och 6. Vi måste alltså ställa om utskriftsläget, och det som passar vårt syfte bäst är fixed.

priser_3.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    double priser[]={0.50,15.50,13};
    double dd=18;
/*Observera tillägget av fixed I utskriften */
    for(int i=0;i<sizeof(priser)/sizeof(double);i++)
     cout<<"Pris: "<<fixed<<setprecision(2)<<priser[i]<<":kr"<<endl;

    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet get nu följande utskrift:

```
Pris: 0.50:kr
Pris: 15.50:kr
Pris: 13.00:kr
Tryck på en valfri tangent för att fortsätta...
```

Så där ja, nu ser det betydligt prydligare ut.

Du kanske inte har tänkt på det, men alla utskrifter är vänsterjusterade, om man inte anger något annat. Vidare är det så, att namnet Morgan skrivs ut med bredden 6 tecken, medan Ove skrivs ut med 4 tecken. Men tänk, om man skulle vilja skriva ut flera namn, där alla namn ska skrivas ut med samma bredd. Det är ju inte alls orealistiskt, man kanske skulle vilja ha en utskrift i kolumner. Då finns en funktion, som heter setw, som tar ett heltal som argument. Setw betyder förstås setwidth och argumentet anger bredden eller vidden på utskriften. Den här funktionen har dock en liten egenhet, som man måste vara vaksam på. Den måste anropas före varje utskrift, för bredden på utskriftslängden går omedelbart tillbaka till defaultläget efter en utskrift.

namn_1.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    const int antal=4;
    string names[antal]={"Ove","Magnus","Morgan","Emil"};
    for(int i=0;i<antal;i++)
        cout<<setw(20)<<right<<names[i]<<endl;
   system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
                 Ove
                Magnus
                Morgan
                 Emil
Tryck på en valfri tangent för att fortsätta...
```

Till vänster om namnen, så ligger precis så många mellanslag, som behövs för att fylla ut bredden 20 tecken. Man kan välja att använda ett annat utfyllnadstecken än tomma tecken. Då använder man funktionen setfill. Den tar en char som argument. setfill('*') fyller således ut med stjärnor.

namn_2.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    const int antal=4;
    string names[antal]={"Ove","Magnus","Morgan","Emil"};
```

```
 /*Observera tillägget av setfill('*') nedan */
    for(int i=0;i<antal;i++)
        cout<< setfill('*')<<setw(20)<<right<<names[i]<<endl;
   system("PAUSE");
    return EXIT_SUCCESS;
}
```

Nu får vi följande utskrift:

```
**********Ove
**********Magnus
**********Morgan
**********Emil
Tryck på en valfri tangent för att fortsätta...
```

Jag ska naturligtvis också visa, hur man kan göra, om man skulle vilja skriva ut förnamn och efternamn på ett antal personer på ett prydligt sätt i två kolumner. Jag döper om arrayen namn till fnamn. Sedan skapar jag ytterligare en array, som jag kallar enam. Utskriften vänsterställs och anropet till setfill tas bort. Bredden på utskriften av förnamn ställs till 20 tecken. Det innebär att utskriften av efternamnet ligger 20 tecken till höger om förnamnet.

namn_3.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
using namespace std;
int main()
{
    const int antal=4;
    string fnamn[antal]={"Ove","Magnus","Morgan","Emil"};
    string enamn[antal]={"Augustsson","Ohlin",
                         "Forsberg","Svedelid"};
    for(int i=0;i<antal;i++)
    {
        cout<<left<<setw(20)<<fnamn[i];
        cout<<enamn[i]<<endl;
```

```
    }
   system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger nu följande utskrift:

```
Ove                 Augustsson
Magnus              Ohlin
Morgan              Forsberg
Emil                Svedelid
Tryck på en valfri tangent för att fortsätta...
```

Prydligt eller hur? Jag tror inte att det skulle vara så svårt för dig att lägga till ytterligare fält till höger om efternamnet, t.ex. telefon.

Vi ska kika på ytterligare ett exempel på formatering av utskrifter. Normalt sett så skrivs det aldrig ut något plustecken framför positiva tal. Om man skulle vilja ha + framför positiva tal, så använder man showpos. Programmet nedan skriver ut ett antal poäng av datatypen doubles med två decimaler och en utskriftsbredd av 3 tecken. Tal skrivs normalt ut med högerjustering. I programmet nedan, så vänsterställs utskriften. Vidare så skrivs + ut framför positiva tal.

positiva_tal.cpp

```
#include <cstdlib>
#include <iostream>
#include<iomanip>
#include<iodos.h>
using namespace std;
int main()
{
    dos_console();
    double scores[]={34.5,-12,456,-15.5};
    int storlek = sizeof(scores)/sizeof(double);
    for(int i=0;i<storlek;i++)
    {
        cout<<"Poäng: "<<setprecision(2)<<fixed<<showpos<<
        showpoint<<setw(4)<<left<<scores[i]<<endl;
```

```
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
Poäng: +34.50
Poäng: -12.00
Poäng: +456.00
Poäng: -15.50
Tryck på en valfri tangent för att fortsätta...
```

Svenska bokstäver i Windows kommandotolk

Det är inte utan att man till och från, åtminstone i datasammanhang, känner sig en aning diskrimimerad som svensk, eller kanske mera korrekt, icke engelsktalande. Vi svenskar använder ju bokstäverna Åå, Ää och Öö. Dessa fick inte plats i den första ASCII-tabellen, ASCII-7. När sedan ASCII-8 tillkom, så fanns det plats för ytterligare 127 tecken.

Det finns olika teckekodningar, men de flesta bygger på att de första 128 tecknen överensstämmer med ASCI-7, medan resterande tecken fram till 255 används på lite olika sätt, t.ex. teckekodningen LATIN_1, som stämmer med det svenska språket.

I svenska installationer av Windows används LATIN_1, utom i ett fall, kommandotolken. Kommandotolken använder som regel en teckenkodning, som heter CodePage 850 och där stämmer bara de första 128 tecknen överens med LATIN_1.

Vi har i boken använt Professor Jan Skansholms förnämliga lösning i iodos.h. Men nu ska vi försöka lösa problemet på egen hand.

Det bör påpekas, att nedanstående lösningar har sina brister, bland annat så är de inte tillräckligt generella. En mer generell lösning hade med all säkerhet inbegripit skapandet av en eller flera klasser, vilket hade fallit utanför ramarna för den här boken. För den intresserade finns säkert en del att hämta på nätet. Själv kan jag varmt rekommendera JanSkansholms klass alpha. Det sagda innebär inte att genomgångarna och exemplen nedan skulle vara ointressanta. Tvärtom så är de mycket lärorika, då de ytterligare belyser hur tecken och teckentabeller fungerar.

Vi gör ett program, som visar vilka talvärden våra svenska bokstäver har i kommandotolken.

tecken_test_1.cpp

```
#include <cstdlib>
#include <iostream>
```

```
using namespace std;
int main()
{
    char tecken[]="åÅäÄöÖ";
    for(int i=0;i<strlen(tecken);i++)
            cout<<tecken[i]<<" = "<<(int)tecken[i]<<endl;
          system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet ger följande utskrift:

```
Õ = -27
┼ = -59
õ = -28
─ = -60
÷ = -10
Í = -42
Tryck på en valfri tangent för att fortsätta...
```

Det är verkligen inte lätt att se vad som har skrivits ut, men eftersom vi själva har skrivit koden, så kan vi ju dra vissa slutsatser. Bland annat så inträffar det häpnadsväckande att talvärdena är negativa. Men vi kan i alla fall skapa en tabell, som visar bokstäverna och deras talvärden.

Bokstav	Talvärde
å	-27
ä	-28
ö	-10
Å	-59
Ä	-60
Ö	-42

Om vi nu gör ytterligare ett program, där vi skriver ut tecknen från 128 till 255, så kanske vi kan hitta de bokstäver vi vill ha. Samtidigt, så skulle vi kunna passa på att skriva ut deras talvärden.

tecken_test_2.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
```

```
int main()
{
  for(int i=128;i<256;i++)
            cout<<(char)i<<" = "<<i<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Programmet gav en väldigt lång utskrift. Av utrymmesskäl visar jag inte hela. Men jag hittade de eftersökta svenska bokstäverna. Hur de såg kan du se här:

```
å = 134
ä = 132
ö = 148
Å= 143
Ä = 142
Ö = 153
```

Till synes ingen ordning på någonting.

Vi skulle ju kunna skapa en funktion, som tar en pekare till char som argument. I funktionen så går vi igenom tecknen i argumentet och kollar upp deras talvärden. Skulle talvärdet vara -42 (ö), så byter vi helt enkelt ut det mot tecknet, som ligger på plats 153. Sedan gör vi likadant med de andra svenska bokstäverna. Vi gör ett försök.

svenska_1.cpp

```
#include <cstdlib>
#include <iostream>
using namespace std;
void svenska(char* str);

int main(int argc, char *argv[])
{
     char str[]="åäöÅÄÖ";
     svenska(str);
     cout<<str<<endl;
```

```
     system("pause");
     return EXIT_SUCCESS;
}

void svenska(char* str)
{
int bort_tecken[]={-27,-28,-10,-59,-60,-42};
int till_tecken[]={134, 132, 148, 143, 142, 153};
int storlek = sizeof(bort_tecken)/sizeof(int);
    for(int i=0;i<strlen(str);i++)
    {
        for(int j=0;j<storlek;j++)
        {
            if( (int)str[i]==bort_tecken[j])
                    str[i]=(char)till_tecken[j];
        }
    }
}
```

Programmet gav följande utskrift:

```
åäöÅÄÖ

Tryck på en valfri tangent för att fortsätta...
```

Glädjande nog så kan vi se, att vi har fått ut svenska bokstäver. I funktionen svenska, så har jag bytt ut felaktiga tecken mot korrekta, genom att konvertera char till heltal, undersöka heltalens värden och skifta till korrekta heltal, då det har varit nödvändigt, innan jag slutligen konverterat heltal till char igen. Vi ska nu pröva en annan variant.

Jag har tidigare i boken tagit upp flyktsekvenser. Vi vet till exempel, att om vi skriver cout<<”\n”;, så får vi ett radbryt. Backslashen anger att tecknet till höger inte är något vanligt tecken. Vi såg ovan att bokstaven å skulle ha talvärdet 134, för att skrivas ut korrekt på skärmen. Om man gör om 134 till ett hexadecimalt format, så får man 86. När man vill använda det hexademicala formatet i C++, så skriver man ett x framför. Det hexadecimala värdet x86 betyder alltså 134. Om man skriver cout<<’\x86’; så skrivs ett vackert å ut på skärmen. Innan jag fortsätter med tecken och teckenkodning, så ska vi göra en liten behövlig avstickare till hexadecimala tal.

Hexadecimala tal

Hexadecimala tal är tal med 16 som talbas. Om man skulle räkna på hexadecimalt sätt, så ser det ut så här: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A, B, C, D, E och F. Man börjar alltså på samma sätt som i det decimala talsystemet, men när man kommer till tio, så börjar man bokstäverna i alfabetet från A till F, som betyder 15. Om du har sysslat något med webbdesign, så har du säkert sett hexadecimala färgvärden. Färgen vit uttrycks med ”#FFFFFF”. I det här fallet, så är det i själva verket tre tal, som sitter ihop, 255 255 255. FF är samma som 255. Hur kan det bli det så?

Jo, precis som i det decimala talsystemet såväl som det binära talsystemet, så är det talens position, som avgöra deras värde. Talet 11 är samma sak som $1 + (1 * 10^{1)}$. Talet 122 är samma sak som $1 + (2 * 10^1) + (1 * 10^2)$. Det där är så självklart att vi inte ens tänker på till vardags. Det är precis på samma sätt med hexadecimala tal tal, skillnaden är bara att vi multiplicerar med 16 istället för 10. Talet 86 är samma sak som $6 + (8 * 16^1)$, vilket get 134. Talet FF blir således $15 + (15 * 16^1)$, vilket blir 255.

Om du har en kalkylator på din dator, så kan den vara utrustat med hexadecimalt talläge. Då kan du pröva att skriva in ett tal i decimalt läge och sedan välja hexadecimalt läge. Då får du veta, hur den hexadecimala representationen av talet ser ut.

Tillbaka till svenska bokstäver i Windows kommandotolk

De talvärden och tillhörande bokstäver vi vill ha ser ut så här:

å = 134 = x86

ä = 132 = x84

ö = 148 = x94

Å= 142 = x8F

Ä = 143 = x8E

Ö = 153 = x99

Genom att använda min kalkylator, så kunde jag ta reda på den hexadecimala representationen.

Då ska vi modifiera funktionen svenska en aning. Nyheterna är fetmarkerade.

```
void svenska(char* str)
{
int bort_tecken[]={-27,-28,-10,-59,-60,-42};char
till_tecken[]={'\x86','\x84','\x94','\x8F','\x8E','\x99'};
int storlek = sizeof(bort_tecken)/sizeof(int);
    for(int i=0;i<strlen(str);i++)
    {
        for(int j=0;j<storlek;j++)
```

```
        {
            if((int)str[i]==bort_tecken[j])
                    str[i]=till_tecken[j];
        }
    }
}
```

Vi testar med ett program.

svenska_2.cpp

```
#include <cstdlib>
#include <iostream>
#include <cstdlib>
#include <iostream>
using namespace std;
void svenska(char* str);

int main(int argc, char *argv[])
{
char str[]="Åke ökar, Ärland är arg åker till Örebro";
     svenska(str);
     cout<<str<<endl;
     system("pause");
     return EXIT_SUCCESS;
}

void svenska(char* str)
{
int bort_tecken[]={-27,-28,-10,-59,-60,-42};
char till_tecken[]={'\x86','\x84','\x94','\x8F','\x8E','\x99'};
int storlek = sizeof(bort_tecken)/sizeof(int);
    for(int i=0;i<strlen(str);i++)
    {
```

```
        for(int j=0;j<storlek;j++)
        {
            if((int)str[i]==bort_tecken[j])
                    str[i]=till_tecken[j];
        }
    }
}
```

Programmet ger nu följande utskrift:

```
ääö ÅÄÖ Åke ökar, Ärland är arg åker till Örebro

Tryck på en valfri tangent för att fortsätta...
```

Sortering med svenska bokstäver i Windows kommandotolk

Om man undersöker talvärdena på de svenska bokstäverna i CodePage 850, så slås man av en sak, nämligen att lilla ä har ett lägre värde än lilla å och att stora Ä har ett värde lägre än stora Å.

Om man utgår ifrån att tecknens talvärde används vid en sortering, så kommer Ärland att hamna före Åke. Vi testart med ett program. Vi kan gott återanvända funktionen vi använde i kapitlet, Att sortera data – sorteringsalgoritmer.

sortera_svenskt_1.cpp

```
#include <cstdlib>
#include <iostream>
#include <cstdlib>
#include <iostream>
using namespace std;
void svenska(char* str);
void sort(string n[],int a);

int main(int argc, char *argv[])
{
    const int ANTAL_NAMN = 4;
    string namn[ANTAL_NAMN]={"Åke","Arne","Ärland","Örjan"};
    /* Vi börjar med att göra om namnen
       till svenska och skriva ut dem på
```

```
       skärmen. Vi måste göra om våra
       strängar till pekare till char.
       Funktionen c_str returnerar en CSträng.
    */
    for(int i=0;i<ANTAL_NAMN;i++)
    {
        svenska((char*)namn[i].c_str());
        cout<<namn[i]<<endl;
    }

    /*Sedan sorterar vi och skriver ut igen.*/
    sort(namn,ANTAL_NAMN);
    cout<<"Efter sorteringen**************"<<endl;
    for(int i=0;i<ANTAL_NAMN;i++)
    {
        svenska((char*)namn[i].c_str());
        cout<<namn[i]<<endl;
    }

     system("pause");
     return EXIT_SUCCESS;
}

void svenska(char* str)
{
int bort_tecken[]={-27,-28,-10,-59,-60,-42};
char till_tecken[]={'\x86','\x84','\x94','\x8F','\x8E','\x99'};
int storlek = sizeof(bort_tecken)/sizeof(int);
    for(int i=0;i<strlen(str);i++)
    {
        for(int j=0;j<storlek;j++)
```

```
        {
           if((int)str[i]==bort_tecken[j])
           {
                str[i]=till_tecken[j];
           }
        }
    }
}
void sort(string n[],int a)
{
string temp;
    for(int i=0;i<a;i++)
    {
        for(int j=0;j<a-1;j++)
        {
            if(n[j]> n[j+1])
            {
                temp = n[j];
                n[j] = n[j+1];
                n[j+1] = temp;
            }
        }
    }
}
```

Programmet ger följande utskrift:

```
Åke
Arne
Ärland
Örjan
Efter sorteringen**********
Arne
```

```
Ärland
Åke
Örjan
Tryck på en valfri tangent för att fortsätta...
```

Jo, det blev väl lite som fruktat. Om jag inte minns fel från min skoltid, så ska Ärland komma efter Åke i alfabetet. Men vi vet ju orsaken, lilla ä har ett lägre talvärde än lilla å och stora Ä har ett talvärde lägre än stora Å. Nu är frågan, vad kan vi göra åt saken?

Ett förslag är att vi kodar om texterna i funktionen sort , alldeles innan vi utför sorteringen. När sorteringen är klar, så kodar vi om texterna igen. Det gäller alltså att flytta tecknen ä och Ä till positioner, som ger dem lägre värden än å och Å. Det är inte en riskfri operation, eftersom vi kan råka hitta positioner, som upptas av andra viktiga tecken. Lilla å har positionen 134. Önskvärt vore om vi kunde tillfälligt kunde göra om lilla ä till ett tecken, som ligger alldeles efter, dvs. 135 Där ligger ett tecken, som ser ut så här ç. Det är ett tecken som inte förekommer i svenska alfabetet, så det kanske inte utgör så stor risk, om vi lånar den positionen. Stora Å har positionen 143. Vi skulle då tillfälligt behöva låna positionen alldeles efter, dvs. 144. Där ligger tecknet É och det är inte heller särskilt vanligt förekommande, så vi prövar att låna den positionen. Vi skriver en helt ny funktion, som vi kallar för sortera_svenska. Då var det dags för lite kod igen. Tilläggas bör kanske, att jag valt att lägga funktionerna i samma fil som programmet och att det egentligen inte är någon bra idé framgår med all önskvärd tydlighet, när det blir mycket kod. Men jag får ursäkta mig med att jag enbart testar mina idéer i det här programmet.

sortera_svenskt_2.cpp

```
#include <cstdlib>
#include <iostream>
#include <cstdlib>
#include <iostream>
using namespace std;
void svenska(char* str);
void sortera_svenskt(string n[],int a);
int main(int argc, char *argv[])
{
    const int ANTAL_NAMN = 4;
    string namn[ANTAL_NAMN]={"Åke","Arne","Ärland","Örjan"};
    /* Vi börjar med att göra om namnen
       till svenska och skriva ut dem på
       skärmen. Vi måste göra om våra
```

```
       strängar till pekare till char.
       Funktionen c_str returnerar en CSträng.
    */
    for(int i=0;i<ANTAL_NAMN;i++)
    {
        svenska((char*)namn[i].c_str());
        cout<<namn[i]<<endl;
    }
    /*Sedan sorterar vi och skriver ut igen.*/
    sortera_svenskt(namn,ANTAL_NAMN);
    cout<<"Efter sorteringen**************"<<endl;
    for(int i=0;i<ANTAL_NAMN;i++)
    {
        svenska((char*)namn[i].c_str());
        cout<<namn[i]<<endl;
    }

     system("pause");
     return EXIT_SUCCESS;
}

void svenska(char* str)
{
int bort_tecken[]={-27,-28,-10,-59,-60,-42};
char till_tecken[]={'\x86','\x84','\x94','\x8F','\x8E','\x99'};
int storlek = sizeof(bort_tecken)/sizeof(int);
    for(int i=0;i<strlen(str);i++)
    {
        for(int j=0;j<storlek;j++)
        {
           if((int)str[i]==bort_tecken[j])
```

```
            {
                str[i]=till_tecken[j];
            }
        }
    }
}
void sortera_svenskt(string n[],int a)
{
string temp;
    /* För flyttar vi på positionerna för
       ä och Ä.*/
    for(int i=0;i<a;i++)
    {
        /* Först letar vi stora Ä och byter ut det. */
        if(n[i].find('\x8e') != string::npos)
        {
            for(int j=0;j<n[i].size();j++)
            {
                if(n[i][j] == '\x8e')
                        n[i][j] = (char) 144;
            }
        }
        /*Sedan letar vi lilla ä och byter ut det. */
        if(n[i].find('\x84') != string::npos)
        {
            for(int j=0;j<n[i].size();j++)
            {
                if(n[i][j] == '\x84')
                        n[i][j] = (char) 135;
            }
        }
```

```
}

/* Sedan sorterar vi som vanligt.*/
for(int i=0;i<a;i++)
{
    for(int j=0;j<a-1;j++)
    {
        if(n[j]> n[j+1])
        {
            temp = n[j];
            n[j] = n[j+1];
            n[j+1] = temp;
        }
    }
}
/* Till sist, så lägger vi tillbaka
   ä och Ä. */

for(int i=0;i<a;i++)
{
    /* Först letar vi stora Ä och byter ut det. */
    if(n[i].find('\x90') != string::npos)
    {
        for(int j=0;j<n[i].size();j++)
        {
            if(n[i][j] == '\x90')
                    n[i][j] = (char) 142;
        }
    }
    /*Sedan letar vi lilla ä och byter ut det. */
    if(n[i].find('\x87') != string::npos)
```

```
        {
            for(int j=0;j<n[i].size();j++)
            {
                if(n[i][j] == '\x87')
                        n[i][j] = (char) 132;
            }
        }
    }
}
```

Programmet ger nu följande utskrift:

```
Åke
Arne
Ärland
Örjan
Efter sorteringen**************
Arne
Åke
Ärland
Örjan
Tryck på en valfri tangent för att fortsätta...
```

Utskriften ger besked om att det fungerade den här gången. Förmodligen så skulle man behöva utföra ganska många test för att känna sig helt säker. Men du har ju redan lärt dig att det är viktigt att testa sina program.

Sammanfattning Sortering med svenska bokstäver i Windows kommandotolk

Det är här ett väldigt komplext område. Men kunskaper om teckenkodning och teckentabeller ingår i en programmerares verktygslåda. Om man söker lite på nätet, så kan man hitta mängder med information i ämnet. Samtidigt som det är besvärligt, så kan det var oerhört fascinerande och intressant att arbeta med. Man känner att man arbetar väldigt nära datorns grundfunktioner och det är spännande.

Det är också så, att många gånger så innebär programmeringsarbetet just bearbetning av text och tecken. Så, det gället att inte ge upp, även om det ibland kan vara frustrerande.

Övningsuppgifter

Addition

Skapa ett program som adderar två decimaltal. Användaren ska först bli tillfrågad, vilka tal denne vill addera. Kontrollera att användaren skriver in giltiga tal. Presentera resultatet på skärmen.

Division

Fortsättning från föregående. Läs nu istället in två decimaltal till två variabler av typen double. Dividera de två talen med varandra och presentera resultatet på skärmen.

Användarens namn

Skapa ett program, som frågar användaren efter förnamn och efternamn. Skriv ut resultatet på skärmen.

Pension snart

I Sverige går man normalt sett i pension, när man fyller 65. Skapa ett program, som frågar efter användarens ålder. Säkerställ att användaren matar in ett korrekt heltal samt att detta är rimligt. Skriv ut på skärmen, hur många år användaren måste vänta på sin pension.

Rabatt

Skapa ett program som räknar ut priset på en rabatterad vara enligt följande exempel:

```
Pris före rabatt: 250.50

Rabatten i procent: 20

Pris efter rabatt: 200.40 Kr
```

Observera att formateringen ska vara som ovan.

Moms

Skapa ett momsberäkningsprogram enligt följande exempel:

```
Pris exkl. moms: 100 kr

Pris inkl. moms: 125 kr
```

Momspåslaget ska alltså vara 25 %. Använd gärna en konstant för momsen.

Ränta på banken

Om man sparar pengar på banken, så erhåller man ränta, inte så mycket, men i alla fall. Nu ska du skapa ett program, som hjälper användaren att beräkna, hur mycket pengar som finns på banken efter viss tid och ränta, som anges av användaren. Programmet kan se ut enligt följande:

```
Vilket år sätter/satte du in pengarna? 2010

Hur många år tänker du spara pengarna? 9

Hur mycket pengar vill du sätta in? 100
```

```
Vad är räntan idag i procent: 5
**************************
Årets ränta: 5.00
År: 2011 Summa: 105.00
**************************
Årets ränta: 5.25
År: 2012 Summa: 110.25
**************************
Årets ränta: 5.51
År: 2013 Summa: 115.76
**************************
Årets ränta: 5.79
År: 2014 Summa: 121.55
**************************
Årets ränta: 6.08
År: 2015 Summa: 127.63
**************************
Årets ränta: 6.38
År: 2016 Summa: 134.01
**************************
Årets ränta: 6.70
År: 2017 Summa: 140.71
**************************
Årets ränta: 7.04
År: 2018 Summa: 147.75
**************************
Årets ränta: 7.39
År: 2019 Summa: 155.13
**************************
```

```
Tryck på en valfri tangent för att fortsätta...
```

Timmar minuter sekunder

Skapa ett program som läser in tid angivit i sekunder och sedan räknar ut:

```
Antal timmar:

Antal minuter:

Antal sekunder:

Exempel:

Ange antal sekunder du vill omvandla: 12892

Antal timmar: 3

Antal minuter: 34

Antal sekunder: 52
```

If

Skapa ett program som lagrar ett tecken. Om tecknet är ett P ska följande skrivas ut: ”Du skrev in ett P”. Om tecknet är ett K ska följande skrivas ut: ”Du skrev in ett K”. Om användaren däremot inte skriver in varken ett P eller ett K ska följande meddelande visas: ”Du skrev inte in ett K eller ett P”.

Exempel på programkörning:

```
Skriv in ett P eller ett K: K

Du skrev in ett K
```

If else

Skapa ett program, som frågar användaren efter temperaturen i rummet denne befinner sig i. Beroende på temperatur, så ska olika meddelanden skrivas ut på skärmen.

Om det är under 18 grader, så uppmanas användaren att ta på sig en tröja.

Om det är mellan 18 och 24 grader, så skrivs, ”Temperaturen är lagom”, ut.

Om det är över 24 grader men under 32, så uppmanas användaren, att öppna ett fönster.

Är temperaturen över 31 grader, så uppmanas användare att kontrollera om det brinner någonstans i rummet eller angränsande rum.

Knop

Skapa ett program som producerar en rapport enligt följande:

Beräkningsprogram för att omvandla knop till km/h

Tabell for knop till km/h

_____

Knop Km/h

1.85

3.70

5.56

7.41

9.26

Formel för beräkning: km/h = 1.852 knop

Användaren ska kunna ange slutvärdet för beräkningen.

Programmet ska gå att köra flera gånger om användaren önskar detta:

Exempel: Vill du köra programmet igen (j/n):

Exempel på programkörning:

```
Ange antal värden du vill beräkna: 8

Tabell för knop till km/h

_____________________

Knop        km/h

1           1.85

2           3.70

3           5.56

4           7.41

5           9.26

6           11.11

7           12.96

8           14.82

Vill du köra programmet igen (j/n)?
```

Stjärnor är vi alla

Skapa ett program som skriver ut en stjärnrad enligt följande:

```
Vilket tecken vill du skriva en stjärnrad av: P

Hur många stjärnor vill du att basen ska h? 20

PPPPPPPPPPPPPPPPPPPP

PPPPPPPPPPPPPPPPPPP

PPPPPPPPPPPPPPPPPP

PPPPPPPPPPPPPPPPP

PPPPPPPPPPPPPPPP

PPPPPPPPPPPPPPP

PPPPPPPPPPPPPP

PPPPPPPPPPPPP

PPPPPPPPPPPP

PPPPPPPPPPP

PPPPPPPPPP

PPPPPPPPP

PPPPPPP

PPPPPP

PPPPP

PPPP

PPP

PP

P
```

Användarens array

Skriv ett program som tar emot 10 heltal från användaren och lagrar dessa i en array. Programmet ska sedan räkna ut summan av de inmatade talen och skriva ut resultatet på skärmen.

Bakvänd array

Skapa ett program som läser in 15 realtal (decimaltal) till en array. Talen ska sedan skrivas ut i omvänd ordning.

Ditt namn baklänges

Skapa ett program som skriver ut ditt namn baklänges enligt följande:

```
Ange antal bokstäver i ditt namn: 7

Skriv in bokstav nr 1 : F

Skriv in bokstav nr 2 : R

Skriv in bokstav nr 3 : E

Skriv in bokstav nr 4 : D

Skriv in bokstav nr 5 :  R

Skriv in bokstav nr 6 : I

Skriv in bokstav nr 7 : K

Ditt namn blir baklänges: KIRDEF
```

Du ska använda en array för att lösa uppgiften!

Medeltemperaturen

Skapa ett program som läser in en veckas temperaturer angivit i Celsius. När 7 dagars temperaturer har angetts ska medeltemperaturen skrivas ut. I lösningen ska en array användas. För att göra det lite mer svårt att starta programmet ska ett lösenord anges för att få starta att skriva in temperaturer. Om fel lösenord används ska programmet avslutas. Lägg dessutom till möjlighet för användaren att köra fler gånger om denne önskar.

Personer

Skapa ett program som lagrar följande om flera personer:

- Klass
- Kön (m/k)
- Skostorlek
- Personnummer (räknare)

Skapa en struktur, som du sedan skapar en array av. Programmet kan se ut enligt följande:

```
Personnummer: 1

Ange klass: IT1A

Ange kön (m/k): m

Ange skostorlek: 44
```

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

Personnummer: 2

Ange klass: IT1B

Ange kön (m/k): k

Ange skostorlek: 36

OSV

Killar har i genomsnitt skostorlek: 44.67

Tjejer har i genomsnitt skostorlkek: 37.00
```

När du matat in uppgifterna i din array ska programmet räkna ut medelskostorleken för killar och för tjejer. Se till att du matar in både killar och tjejer annars kan det bli division med noll. Vidare så skulle det kunna vara av intresse att se, vilken klass, som har störst fötter.

Parameterlös funktion

Skapa ett program med en parameterlös procedur som skapar en rad med stjärnor enligt följande:

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

Funktion med parameter

Bygg ut föregående övning, så att användaren kan välja hur många tecken med stjärnor som ska skrivas ut. Proceduren måste nu ta en parameter. Exempel:

```
Hur många tecken vill du skriva ut: 34

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

Funktion med två parametrar

Skapa en funktion, som har två parametrar. Den ena ska ange vilket tecken du vill skriva ut och den andra ska ange hur många gånger du vill skriva ut tecknet. Det är alltså användaren som ska bestämma vad som ska skrivas ut på skärmen.

Negativ eller positiv?

Skapa ett program med en funktion. Funktionen ska vara boolsk och kolla om användaren har matat in ett positivt eller negativt tal. Funktionen ska ha en parameter av heltalstyp. Exempel:

```
Skriv in ett positivt eller negativt tal: -56

Talet är negativt
```

Norska kronor

Skapa ett program som konverterar svenska kronor till norska kronor. För 100 svenska kronor får du 89 norska kronor. Använd en funktion för att lösa uppgiften. Exempel:

```
Skriv in antal svenska kronor du vill räkna om till norska: 200

Det blir i norska kronor: 178.00
```

Celsius och Fahrenheit

Celsius och Fahrenheit är två system för att uttrycka temperaturer. Gör ett program, som innehåller de två funktionerna celsuisToFahrenheit och fahrenheitToCelsius. Funktionen celsuisToFahrenheit tar en double (Celsius) som argument och returnerar en double (Fahrenheit). fahrenheitToCelsius gör samma sak fast tvärtom. Formlerna ser ut som följer:
celsius = (fahrenheit-32)*5/9
fahrenheit = celsius * 9/5 +32

Avdragsberäkning

Enligt svensk skattelagstiftning, så kan man som arbetstagare dra av för kostnader för resor till och från arbetet med egen bil. Summan man får dra av beräknas per mil. Summan kan ändras från år till år. Förhoppningsvis, så går den upp.

Du ska göra ett program, som räknar ut användarens totala avdrag för resor till och från arbetet med egen bil under ett år. Eftersom summan man får dra av kan variera från ett år till ett annat, så får du be användaren om den upplysningen.

Vilka tecken används mest

Skapa ett program, som öppnar en fil på din hårddisk. Trevligast vore det väl om det handlar om vanlig löptext, kanske en berättelse. Sedan ska programmet ge en utskrift enligt nedan:

```
Tecken och dess förekomst i filen ”Minberättelse.txt”.

Tecken          Förekommer antal gånger

A               89

B               23

osv.
```

Att byta ut text

Skapa en textfil med följande innehåll:

Jag har bott i Hjo i hela mitt liv. Hjo är utan tvekan Sveriges trevligaste stad. Innevånarna i Hjo går omkring med ett lyckligt leende, som visar sig redan i späda barnaår och håller till ålderdomen.

Spara filen som min_hemstad.txt

Skapa ett program, som läser in innehållet i texten och byter ut förekomsten av Hjo mot namnet på din hemstad. Öppna sedan filen min_hemstad.txt och skriv över den gamla texten med den nya.

(Om du händelsevis skulle bo i Hjo, så kan du hoppa över den här övningen (-;)

Lottorader

Lotto är ett ganska populärt spel, som går ut på att välja ut sju nummer av 35, som dras ett par gånger i veckan. Finessen med Lotto är, att det är helt och hållet turbaserat. Det innebär, att den, som lyckas pricka in sju rätt kan påräkna en hög vinst. Jag tänkte att du skulle försöka skapa ett program, som fungerar enligt nedanstående programutskrift.

```
Hur många rader vill du skapa?

10

Vill du ta bort några tal[y/n]: y

Du matar in de siffror du vill ta bort!

Du kan välja bort högst tio nummer.

0 avslutar inmatningen.

10

15

16

17

18

0

Följande rader hittades:

1  |3  |6  |13|24|28|33

13|22|26|27|29|33|34

2  |5  |7  |8  |14|25|29

2  |4  |13|19|24|26|30

7  |9  |14|23|28|33|34

2  |5  |12|14|24|30|35

1  |4  |9  |21|29|31|33

20|23|24|25|26|32|35

6  |11|12|14|21|23|26

4  |12|14|21|25|33|34

Vill du spara raderna till fil?[y/n]
```

Lotto 7 rätt

Testa programmet ovan. Dra ett valfritt antal rader. Sedan gör du ett program, som simulerar lottodragningar. Frågan är, hur många dragningar fordras innan du får 7 rätt.

Miniräknaren

Datorer är väldigt duktiga på matte. Det ska man naturligtvis utnyttja till att skapa en miniräknare. Du kan göra den hur avancerad som helst, men du bör minst ha med de fyra räknesätten.

Man skulle kunna tänka sig att användare kommer till en meny vid uppstarten

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

MENY

****************************************

1 addition

2 subtraktion

3 division

4 multiplikation

5 avsluta
```

Slutord från författaren

Det är roligt att programmera. Det är en konst, som man ständigt kan utvecklas i. Själv har jag programmerat i ganska många år nu, men jag upptäcker ständigt nya härliga utmaningar, så förutom att programmering ingår som en del i mitt jobb, så är programmering också min främsta hobby.

Det är dock så, att det kan te sig ganska så besvärligt och krångligt att lära sig programmering som nybörjare. Jag har i boken försökt använda den metod, som jag genom åren har tillämpat i klassrummet. Den innebär i all sin enkelhet, att man varvar teori och praktik. Det är inte alltid, som den teoretiska grunden finns med i varje praktiskt moment. Tanken är att praktiska resultat ska öka aptiten för teoretiska kunskaper. Det kan gå till på det viset, att man börjar med en liten kortare teoretiskt genomgång, t.ex. vad är ett program? Om eleverna aldrig tidigare har funderat över frågan, så kan resonemanget te sig helt obegripligt eller i alla fall ohyggligt tråkigt. Därför så följs genomgången upp med att eleverna får göra ett program, kanske sitt första rentutav. Då brukar det glimta till i ögonen på eleverna, de tycker att det är roligt. Eleverna behöver inte ens förstå alla detaljerna i programmet, för att tycka att det är spännande att skriva kod, kompilera och provköra. Men var så säker, deras nyfikenhet väcks. Då kan det vara dags att ha nästa lilla teoretiska genomgång, kanske om hur man kan tänka, när man löser enklare problem med någon/några variabler och en selektion. Man kan följa upp med något handfast exempel, som visar, hur man kan lösa ett problem, med en selektion, dvs. if. Låt sedan eleverna själva skapa några små program, som använder en if-sats osv.

När man bedömer elevernas lösningar på t.ex. övningsuppgifter, så är det viktigaste att eleven har lyckats lösa uppgiften, inte hur? Förfining av algoritmer kommer med tiden.

Det är min förhoppning, att du som har läst och studerat boken skolelev, nybörjare eller intresserad i allmänhet tycker att det har varit givande och intressant. Jag hoppas också att jag har väckt och lyckats behålla ditt intresse för programmering. Nu är det dags för dig att gå vidare i din utveckling som programmerare. Nästa steg i den utvecklingen är objektorienterad programutveckling. I objektorienterad programutveckling, så är begreppen klass och objekt centrala. Vi har till och från i den boken berört dessa begrepp. Ja vi har till och med använt en del av de ”färdiga” klasser, som finns i C++ standardbibliotek. Vidare så har vi studerat och använt strukturer, som kan ses som en klass. Det här innebär att du bör vara ganska väl förberedd för nästa steg i din programmering.

Det är visserligen undertecknad, som har skrivit den här boken, men jag bör rikta en tacksamhetens tanke till alla de programmeringsintresserade elever och ungdomar jag mött genom åren. Ungdomar är härliga i sin nyfikenhet och i sitt många gånger okonventionella tankesätt. Alla dessa elever har på ett eller annat sätt bidragit till boken.

Vidare så riktar jag en tacksamhetens tanke till min kollega på it-gymnasiet i Skövde, Fredrik Hasselplan, som har fungerat som samtalspartner och idéspruta. Han har kommit med många kloka råd och uppslag. Ett ganska stort antal av bokens avslutande övningsuppgifter har Fredrik som skapare. Tack så mycket Fredrik!

Två av mina elever, Andreas Gren och Philip Hassel har lämnat ett bidrag i form av en funktion, som skriver ut en hängande gubbe. Denna återfinns i exemplet ”Hänga gubbe”.

Med Vänliga Hälsningar Morgan Augustsson

Hjo 1/11 2010