

---

# Programmering 1

Del 1: Grunderna

Martin Larsson, Andreas Larsson

19 juni 2019

# Innehåll

<b>1</b>	<b>Introduktion</b>	<b>4</b>
<b>2</b>	<b>Om programmering</b>	<b>5</b>
2.1	Kompilerande språk . . . . .	5
2.2	Skriptspråk . . . . .	5
2.3	Att göra . . . . .	5
<b>3</b>	<b>Om algoritmer</b>	<b>6</b>
<b>4</b>	<b>Sekvens</b>	<b>8</b>
4.1	Sekvens i C# . . . . .	9
4.2	Att göra . . . . .	10
4.3	Övningar . . . . .	10
4.4	Variabler, datatyper och in/utmatning . . . . .	10
4.4.1	Vanliga, enkla datatyper . . . . .	11
4.4.2	Variabelnamn . . . . .	11
4.4.3	Deklaration och tilldelning . . . . .	12
4.4.4	Inläsning . . . . .	13
4.4.5	Typomvandling . . . . .	13
4.4.6	Enkla beräkningar . . . . .	14
4.4.7	Utmatning . . . . .	14
4.5	Att göra . . . . .	15
4.6	Övningar . . . . .	15
<b>5</b>	<b>Iteration</b>	<b>17</b>
5.1	Iteration i C# . . . . .	19
5.1.1	For-satsen . . . . .	19
5.1.2	While-satsen . . . . .	21
5.1.3	Enkla jämförelser . . . . .	22
5.2	Att göra . . . . .	22
5.3	Övningar . . . . .	22





<b>6</b>	<b>Selektion</b>	<b>26</b>
6.1	Selektion i C# . . . . .	27
6.1.1	Flera villkor & logiska operatorer . . . . .	30
6.2	Att göra . . . . .	31
6.3	Övningar . . . . .	31
<b>7</b>	<b>Blandade övningar</b>	<b>35</b>
7.1	Villkorsoperatör . . . . .	35
7.2	<i>switch-case</i> -satsen . . . . .	36
7.3	<i>break</i> . . . . .	37
7.4	Slumptal . . . . .	38
7.5	Kommentarer . . . . .	38
7.6	Övningar . . . . .	39

# 1 Introduktion

Detta är ett introduktionsmaterial i programmering. Det är utformat för att användas i kursen Programmering 1 på gymnasiet och de programkodexempel som finns med är skrivna i C#. Dokumentet innehåller information, genomgångar, exempel, övningar och länkar till ytterligare/fördjupande material.

Sammanställningen har gjorts av Andreas Larsson och Martin Larsson, båda lärare på Erik Dahlbergs-gymnasiet i Jönköping.

På flera ställen i dokumentet finns *Att göra*-listor. Dessa markeras med olika symboler för att förtydliga.

-  Läs vidare i den kompletterande kursboken (sidnummer anges): Lilja, M. & Nilsson, U. (2012). *Programmering 1 C#*. Malmö: Gleerups.
-  Studera externt material på webben (länk bifogas). Ofta från [csharpskolan.se](http://csharpskolan.se) men ibland även andra webbplatser.
-  Genomför övningar (i dokumentet, i kursboken eller via extern resurs).
-  Genomför självtest på Moodle: [moodle.teed.se](http://moodle.teed.se).

Lycka till med dina studier!

## 2 Om programmering

Det finns massor av olika språk att programmera med. I Wikipedia-artikeln *Comparison of programming languages* ([en.wikipedia.org/wiki/Comparison\\_of\\_programming\\_languages](https://en.wikipedia.org/wiki/Comparison_of_programming_languages)) finns över 100 st. och då är det ändå bara ett urval av språk. Det finns två huvudsakliga kategorier: kompilerande språk och interpreterande språk (eller skriptspråk).

### 2.1 Kompilerande språk

Kompilerande språk körs genom en kompilator för att skapa exekverbara (körbara) program. Kompilatorn är en ”byggmodul” som översätter programkod till maskinkod, en datoranpassad kod som är mer eller mindre oläslig för människor. Exempel på kompilerande språk är C, C++, C#, Java, Fortran och Pascal. Java och C# är egentligen lite av ett specialfall då dessa kompileras till något som kallas bytekod istället för maskinkod men vi går inte djupare in på det i den här kursen.

### 2.2 Skriptspråk

De språk som inte är kompilerande måste ha någon form av tolkningsprogramvara installerad på datorn för att kunna köras. Det skapas ingen exekverbar fil utan istället körs källkoden in i tolken som sedan genererar resultat. Några exempel på skriptspråk är JavaScript, Perl, PHP och Python.

### 2.3 Att göra

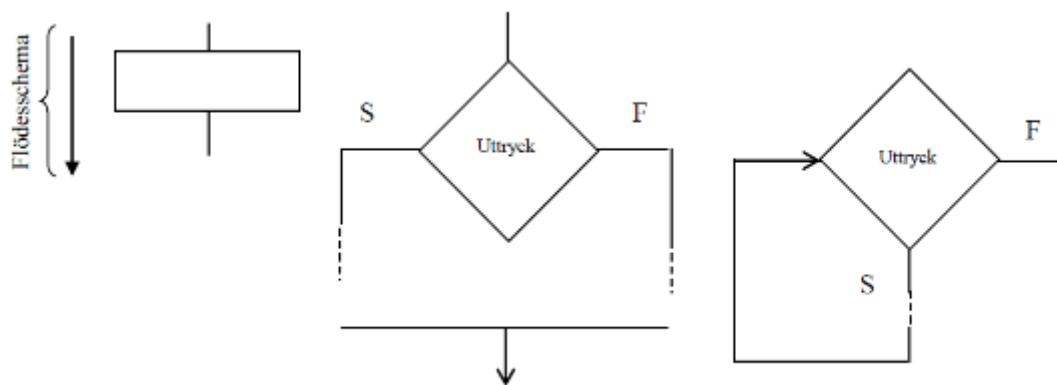
- 📖 Läs mer om programmering generellt i kursboken s. 7-10
- 🌐 Läs igenom artikeln ”Vad är programmering?” på C#-skolan [csharp-skolan.se/article/vad-ar-programmering](https://csharp-skolan.se/article/vad-ar-programmering)

### 3 Om algoritmer

En algoritm är en metod för att lösa ett problem. Metoden består av ett antal enkla instruktioner. Givet förutsättningarna leder metoden efter ett *ändligt* antal instruktioner till ett resultat. Algoritmen beskriver problemlösningen för en *maskin*. Alltså beskriver den verkliga problem med ett konstlat (pseudo) språk. Algoritmbeskrivningen visualiserar en övergripande idé av hur ett program eller maskin skall lösa ett problem. Vardagliga exempel på algoritmbeskrivningar är möbelmonteringsanvisningar eller matrecept.

För att beskriva algoritmer i programmering använder man sig av antingen **pseudokod** eller **strukturdiagram**. Pseudokod är ett språk som kan anses vara ett mellanting mellan ett programspråk och vårt vardagliga tal. Det är formellare än vår svenska men inte lika strikt i sin syntax som ett riktigt programmeringsspråk. Tanken med pseudokod är att det ska ge en god beskrivning av en algoritm oavsett vilket programmeringsspråk den tänkta läsaren är bekant med. Strukturdiagram är en mer visuell representation av en algoritm. Dessa diagram ska ge överblick och hjälpa oss människor att förstå exekveringen av instruktionerna.

Strukturdiagrammen och pseudokoden byggs upp av de tre delarna sekvens, selektion och iteration. Dessa delar kallas **kontrollstrukturer** (mer om dessa i kommande kapitel).



**Figur 3.1:** Sekvens, selektion och iteration som strukturdiagram.

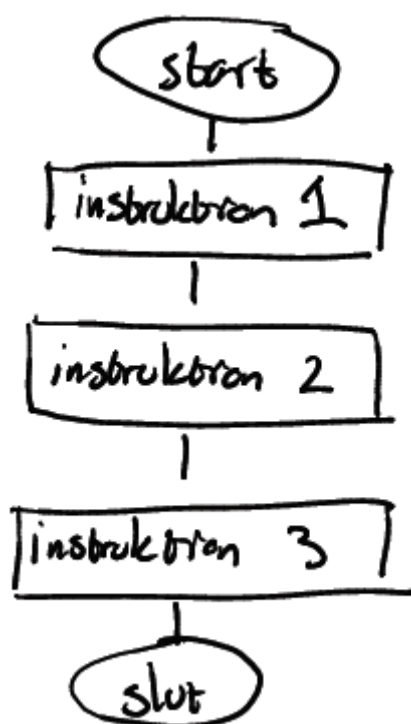
Ett strukturdiagram ritas alltid med en **start** och ett **slut** (i ovala figurer). Sekvenser ritas i boxar och

uttryck i diamantfigurer. Flödet i diagrammet anges med pilar. *S* och *F* i figuren står för sant respektive falskt - vilken väg man skall gå beror på värdet av det uttryck som är aktuellt.

Uttryck skrivs i både pseudokod och strukturdiagram som *om <villkor> så* (selektion) och *så länge <villkor>* (iteration). Det enklaste och snabbaste sättet att skapa strukturdiagram är att använda penna och papper. När man skriver pseudokod är det viktigt att vara noga med indrag (indentering) eftersom dessa visar hur algoritmen fungerar.

Även när man skriver pseudokod finns *start* och *slut* med. Kodstyckena däremellan är indragna ett steg (och ytterligare steg vid iteration och selektion, se kommande kapitel).

```
1 start
2     instruktion 1
3     instruktion 2
4     instruktion 3
5 slut
```



**Figur 3.2:** Exempel på strukturdiagram med tre sekvenser.

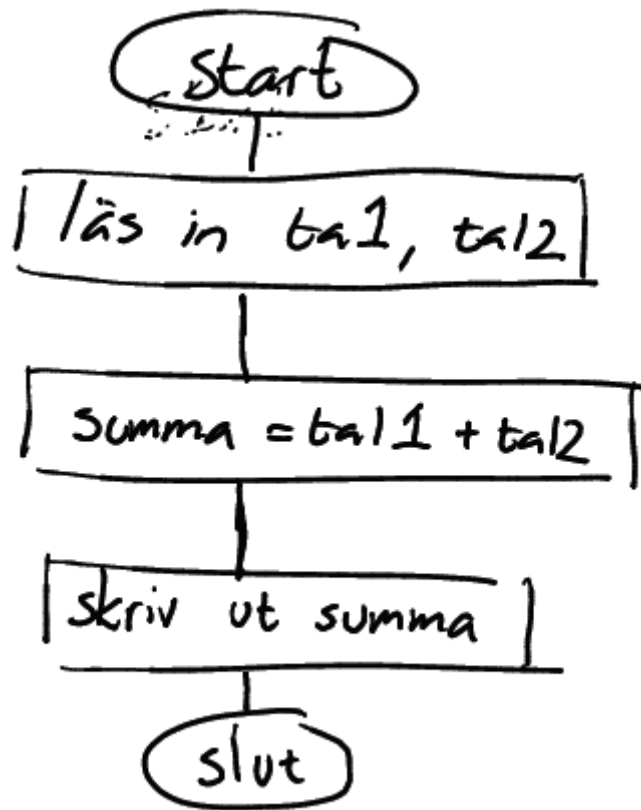
I nästa kapitel tittar vi närmare på att skapa algoritmer utifrån de tre kontrollstrukturerna sekvens, iteration och selektion med hjälp av strukturdiagram, pseudokod och källkod (C#-kod).

## 4 Sekvens

Sekvenser i pseudokod skrivs helt enkelt som instruktioner, rad för rad. I strukturdiagrammen representeras varje rad av en box och dessa förbinds med streck. Till exempel kan algoritmbeskrivningen för att addera två tal se ut såhär:

```
1 start
2   läs in tal1
3   läs in tal2
4   summa = tal1 + tal2
5   skriv ut summa
6 slut
```





**Figur 4.1:** Addera två tal som strukturdiagram.

## 4.1 Sekvens i C#

Som vi ser i exemplet ovan är en sekvens ingen specifik instruktion utan snarare en instruktioner generellt. Motsvarigheten till detta i C# blir då en kodrad (eller ett kodblock). Men för att vi ska bekanta oss med några vanliga, grundläggande C#-sekvenser så tittar vi på ett par specifika sådana här. Tänk på att varje sats skall avslutas med semikolon (;).

- Läs in: `Console.ReadLine();`
- Läs in och spara värde: `var variabelnamn = Console.ReadLine();`
- Skriva ut text: `Console.WriteLine("Texten som ska skrivas");`
- Skriva ut text med avslutande radbrytning: `Console.WriteLine("Texten som ska skrivas");`
- Skriva ut värde av variabel: `Console.WriteLine(variabelnamn);` / `Console.WriteLine(variabelnamn);`

- Tolka ett inmatat värde som heltal: `var tal = int.Parse(inmatatVarde);`

Att genomföra beräkningar med de fyra vanliga operatorerna (+, -, \*, /) är inga problem så länge man har korrekta datatyper. Tänk dock på att tilldelning sker åt vänster i programspråket, t.ex. så genomför datorn följande sats `var kraft = massa * 9.82` i dessa steg:

1. Hämta värdet som finns lagrat i variabeln `massa`.
2. Beräkna högerledet.
3. Tilldela resultatet av beräkningen till variabeln i vänsterledet.

## 4.2 Att göra

- 📖 Läs igenom s. 11-12 i boken.
- 🌐 Bekanta dig med Visual Studio och dess struktur via artikeln *Filstruktur och projekt* på [csharpskolan.se/article/filstruktur-och-projekt](https://csharpskolan.se/article/filstruktur-och-projekt).
- ✍️ Genomför övningarna 1-5 här nedan.
- ✍️ Genomför *Laboration 1 - "Hej världen" med färger* på [csharpskolan.se/article/laboration-1-hej-varlden-med-farger](https://csharpskolan.se/article/laboration-1-hej-varlden-med-farger)

## 4.3 Övningar

Beskriv följande som algoritmer i form av både strukturdiagram och pseudokod. Implementera sedan algoritmerna i C#. Du kan anta att samtliga inlästa tal är heltal.

1. Läs in två tal och beräkna produkten.
2. Läs in två tal, a och b. Skriv sedan ut differensen `a-b` och `b-a`.
3. Omvandla en tidsangivelse i timmar, minuter och sekunder till endast sekunder.
4. Beräkna arean av en kub. Läs in längden på en av kubens sidor och skriv ut arean.
5. Beräkna volymen av ett rätblock. Läs in nödvändiga värden och skriv ut volymen.

## 4.4 Variabler, datatyper och in/utmatning

Nyckelordet `var` som vi hittills använt för variabler är egentligen en speciallösning. Det används när variabeln man tänker använda är av okänd typ eller behöver anpassa sig på något sätt. Om man vet om vilken datatyp man ska hantera (vilket vi nästan alltid gör) så bör man specifikt ange den faktiska datatypen och därmed undvika `var`, även om det faktiskt fungerar.

#### 4.4.1 Vanliga, enkla datatyper

I C# finns en hel del olika datatyper som kan lagra olika data. Några av dem är:

- **short**: heltal mellan -32768 och 32767
- **int**: heltal  $\pm 2 * 10^9$
- **long**: heltal  $\pm 9 * 10^{18}$
- **float**: decimaltal med 7 gällande siffror
- **double**: decimaltal med 15 gällande siffror
- **char**: ett tecken
- **string**: text (sträng av tecken)
- **bool**: **true** eller **false**

Det är dock ganska vanligt att man nöjer sig med att använda **int**, **double**, **char**, **string** och **bool**. På så vis blir det inte så jättemycket att lära sig.

Några saker att tänka på för vissa datatyper i C#:

- I källkoden används punkt som decimaltecken och inte komma.
- Värdet på strängar **string** skall alltid starta och sluta med citattecken (`"`). `string text = "Hej"`; är korrekt men `string text = Hej`; genererar ett fel.
- Tecken omringas av *single quote* (`'`), t.ex: `char tecken = 'a'`;
- Datatypen **bool** kan *endast* lagra värdena **true** eller **false** och dessa ska ej omringas av citattecken.

#### 4.4.2 Variabelnamn

Du kan namnge dina variabler nästan hur du vill men det finns några regler man inte kan bryta mot. Variabelnamn i C# får ej

- Innehålla mellanslag
- Börja på ett tal

Dessutom är det några regler man kan men inte *bör* bryta mot. Man bör

- Låta sina variabelnamn börja på liten bokstav.
- Ha tydliga, beskrivande namn på sina variabler.
- Undvika åäö.
- Ange stor bokstav vid nytt ord, t.ex: `antalPoang` och inte `antalpoang` eller `antal_poang`.

### 4.4.3 Deklaration och tilldelning

När en variabel ”skapas” så kan det genomföras på två olika sätt. Processen delas upp i två viktiga begrepp: **deklaration** och **tilldelning**. Att deklarera en variabel är att säga åt datorn att ”den här variabeln skall finnas och den ska vara av den här typen” men den får inget värde eftersom att vi inte lagrar någonting i den direkt. En deklaration består endast av `datatyp variabelnamn;`. Till exempel:

```
int antal;
```

eller

```
double temperatur;
```

Tills dess att variabeln har tilldelats ett värde (med =) har den ”standardvärdet”/tomma värdet vid namn **null**. Om en variabel är deklarerad och skall tilldelas ett värde ska datatypen ej anges igen. T.ex:

```
antal = 5;
```

eller

```
temperatur = 27.6;
```

Många gånger vet man direkt vid deklarationen av en variabel vilket värde den skall tilldelas. Då kan man slå ihop deklaration och tilldelning på en och samma rad och istället utföra en så kallad **initialisering**. Då ser det helt enkelt ut såhär:

```
int antal = 5;
```

Algoritmen för att addera två tal från förra kapitlet skulle med andra ord kunna se ut på minst två olika sätt i C#:

```
1 string inlasning;  
2 inlasning = Console.ReadLine();  
3 int tal1;  
4 tal1 = int.Parse(inlasning);  
5 inlasning = Console.ReadLine();  
6 int tal2;  
7 tal2 = int.Parse(inlasning);  
8 int summa;  
9 summa = tal1 + tal2;  
10 Console.WriteLine(summa);
```

```
1 string inlasning = Console.ReadLine();  
2 int tal1 = int.Parse(inlasning);  
3 inlasning = Console.ReadLine();
```

```
4 int tal2 = int.Parse(inlasning);
5 int summa = tal1 + tal2;
6 Console.WriteLine(summa);
```

#### 4.4.4 Inläsning

Lägg märke till raden `string inlasning = Console.ReadLine();` i exemplet ovan. Där vi tidigare använde `var` använder vi nu `string` - den egentligen datatypen vid inläsning från konsoll. Resultatet är dock detsamma, en variabel av typen `var` blir en `string` när koden faktiskt körs. Men om vi deklarerar eller initialiserar med `string` från början så har vi bättre koll på våra typer även i resten av källkoden.

Med andra ord kan du från och med nu ta förvana att alltid använda `string` när du skall läsa in från konsollen.

#### 4.4.5 Typomvandling

Typomvandling är egentligen något du redan stött på. Varje gång vi läser in från konsollen har vi fått data i form av en text/sträng (`string`) men har sedan använt den i heltalsberäkningar. Med andra ord är det en typomvandling vi har gjort när vi använt funktionen/metoden `int.Parse()` ;.

För alla datatyperna utom `string`, dvs. `int`, `double`, `char`, `bool` så finns en `.Parse()` -funktion som omvandlar till just den datatypen (om så är möjligt). Det kan med andra ord se ut såhär:

```
1 string inlasning = Console.ReadLine();
2 int heltal = int.Parse(inlasning);
3 double decimaltal = double.Parse(inlasning);
4 char tecken = char.Parse(inlasning);
5 bool santEllerFalskt = bool.Parse(inlasning);
```

Om man istället vill omvandla från en datatyp till en textsträng så finns en funktion `.ToString()` som kan köras på alla variabler. Man skriver helt enkelt bara `variabelnamn.ToString()` ;. Ska man till exempel lagra ett tal som text kan man skriva:

```
1 int tal = 27;
2 string talSomText = tal.ToString();
```

#### 4.4.6 Enkla beräkningar

Precis som tidigare nämnt så är det inga konstigheter med att använda de fyra vanliga räkneoperatörerna. Addition, subtraktion och multiplikation fungerar precis som tänkt. Division däremot kan tyckas bete sig lite underligt om man inte är uppmärksam på heltal och decimaltal. Exempelvis så resulterar den här kodraden i svaret 0.

```
1 Console.WriteLine(4 / 5);
```

Vilket är rätt endast om vi bara tar hänsyn till heltal. Hade vi tänkt oss ett svar i decimalform är det ju felaktigt. För att ett svar skall skrivas i decimalform måste en av siffrorna som ingår i beräkningen skrivas på decimalform. Med andra ord skulle nedanstående rad ge ett annat resultat.

```
1 Console.WriteLine(4 / 5.0);
```

Om resultatet ska lagras i en variabel istället för att skrivas ut direkt måste denna vara av typen **double**.

#### 4.4.7 Utmatning

Hittills har vi gjort utmatningar med endast variabelvärden eller text. Inte både ock. För att åstadkomma detta är det några saker man ska vara noga med att inte göra fel på. Låt säga att vi ska skriva ett program som läser in ålder och sedan skriver ut texten "Min ålder är" tillsammans med siffran man matat in. Då skulle det kunna se ut såhär (koden för inläsning har uteslutits, istället används en variabel för ålder direkt):

```
1 int alder = 18;  
2 Console.WriteLine("Min ålder är: " + alder);
```

Additionstecknet i det här fallet innebär att texten "Min ålder är:" skall slås i hop med värdet av variabeln **alder**. När en text slås ihop med en variabel av annan typ tolkas alltid variabeln om (automatiskt) till textformat (som om man kört `.ToString()` på den) och resultatet blir att en text följs av en annan. Med andra ord kan man tänka sig att det som faktiskt står i kod är:

```
1 Console.WriteLine("Min ålder är: " + "18");
```

Additionstecknet mellan text kallas för **konkatenering**. Ihopslagning av strängar helt enkelt. Om det inte är en sträng med i bilden och det istället handlar om ett additionstecken mellan två **int** i vår **WriteLine**-rad så utförs den faktiska beräkningen och resultatet av denna skrivs ut. Det här kodstycket skulle med andra ord skriva ut 5.

```
1 int tal = 3;
2 Console.WriteLine(tal + 2);
```

Vi har använt `Console.WriteLine()` och `Console.Write()` för att göra utskrifter med respektive utan radbrytning. Om man vill göra en radbrytning mitt i en utskrift (utan att behöva skriva en ny kodrad) kan man använda sig av `\textbackslash n` - dessa två tecken tillsammans tolkas som ett tecken som skrivs ut som radbrytning. Man kallar det för *newline*-tecken men läser ofta ut det som *backslash n*. Vi kan alltså skriva `Console.Write("Erik \textbackslash n Dahlberg")` i C# för att göra en utskrift med *Erik* på en rad och *Dahlberg* på nästa. Ett liknande tecken kan användas för att skriva ut ett indrag (tab). Då använder man istället `\textbackslash t`.

## 4.5 Att göra

- 📖 Mer att läsa om variabler och typomvandlingar finns i boken s. 26-30.
- 🌐 Några ytterligare rader om variabler finns på [csharpskolan.se/article/variabler](http://csharpskolan.se/article/variabler).
- ✎ Genomför övningarna 6-13 här nedan.
- ✅ Genomför självtestet *In/utmatning, variabler och datatyper* på Moodle: [moodle.teed.se](http://moodle.teed.se).

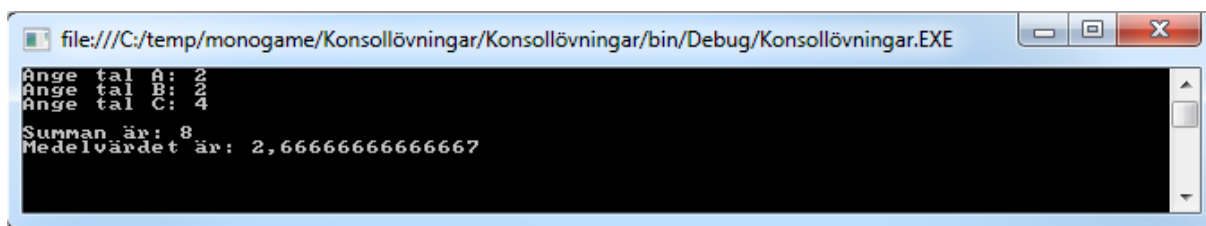
## 4.6 Övningar

Lös följande övningar i tre steg: Strukturdiagram, pseudokod och sedan i C#.

6. Omvandla grader Celsius(läs in) till grader Fahrenheit och skriv ut resultatet. (Formeln är  $F = 1,8 * C + 32$ )
7. Beräkna en cirkels omkrets och area med hjälp av radiens längd.
8. Beräkna en försäljares månadslön. Försäljaren har en grundlön på 15 000 kr i månaden. Dessutom tillkommer 8 % på försäljningssumman(läs in).

Uppgifterna nedan behöver du bara lösa i C#, varken strukturdiagram eller pseudokod behövs.

9. Skriv ett program där man kan mata in tre heltal av typen int och beräkna summan och medelvärdet av talen.



```

file:///C:/temp/monogame/Konsollövningar/Konsollövningar/bin/Debug/Konsollövningar.EXE
Ange tal A: 2
Ange tal B: 3
Ange tal C: 4
Summan är: 8
Medelvärde är: 2,666666666666667
    
```

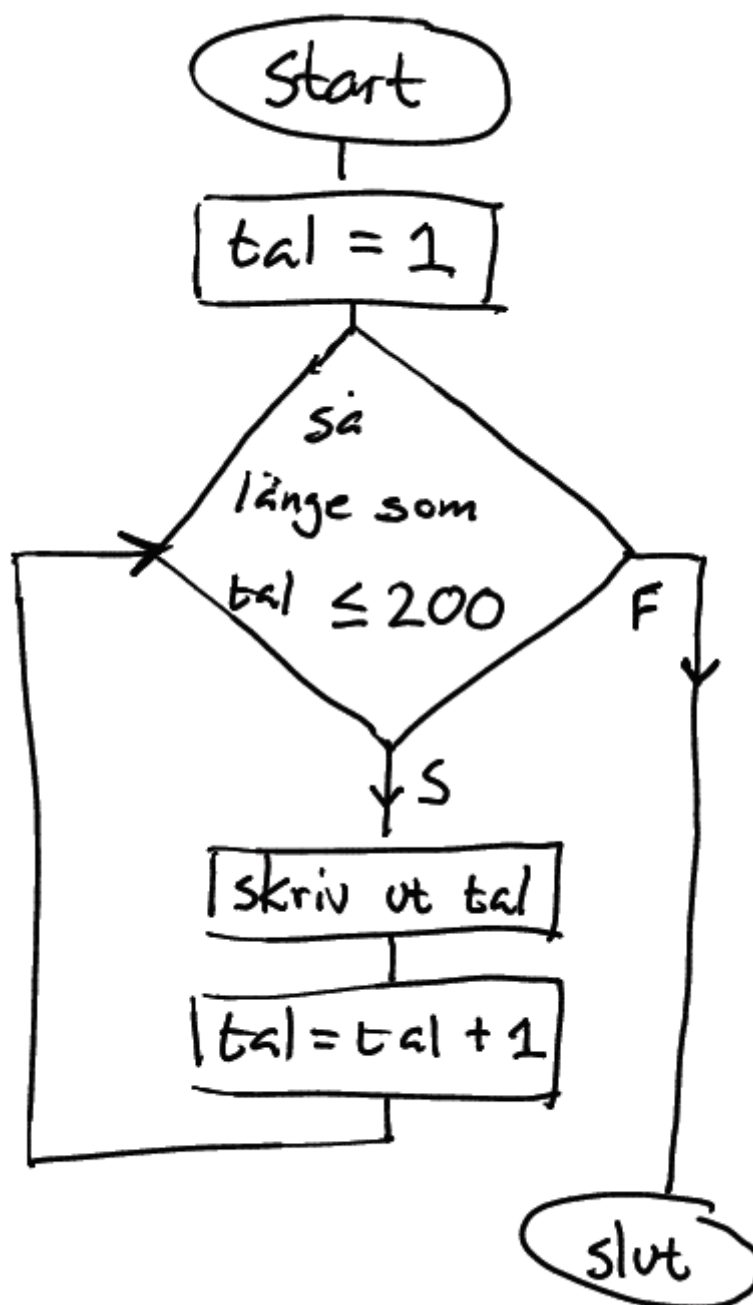
**Figur 4.2:** Körningsexempel för medelvärdesprogrammet.

10. Skriv ett program för en uttagsautomat där man kan skriva in ett belopp, och som visar hur stort uttag man gjorde. Vi antar att automaten bara har hundralappar, så om man skriver till exempel 670 ska det skrivas ut "Uttag: 600 kr".
11. Det blir lite svårare om automaten även har 500-lappar. Utöka programmet i uppgift 10 så att automaten svarar så här istället (om man skriver in 670): Uttag i 500-sedlar: 500 kr. Uttag i 100-sedlar: 100 kr.
12. Du känner förmodligen igen sifferleken nedan. Man ber en kompis tänka på ett tal, addera 1 och så vidare. Till slut talar man om för kompisens vilket tal han kommit fram till, och om det är första gången kompisens hör sifferleken undrar han kanske hur du kunde veta det. Tänk på ett heltal. Addera 1. Multiplicera med 2. Subtrahera 6. Halvera. Addera 3. Subtrahera talet du tänkte på. Nu har du fått talet 1! Gör ett program där man kan mata in ett tal. Provkör med några olika tal. Om du gjort rätt ska resultatet alltid bli 1.
13. Gör ett program som omvandlar ett bråktal till blandad form. Man ska kunna skriva in en täljare och en nämnare och trycka på en knapp. Om användaren till exempel skriver in 5 och 3 ska resultatet visas som "5/3 blir 1 2/3 i blandad form."



## 5 Iteration

Många gånger kan det vara till stor hjälp att något kodstycke upprepas. Det kan t.ex. röra sig om uppräknings av ett tal, flera utskrifter i rad, upprepad inmatning och mycket annat. Vi kallar denna typ av konstruktion för **iteration**. På ren svenska säger vi att något skall göras *så länge som* något är sant. Vi kan titta på det enkla exemplet med att skriva ut alla heltal från och med 1 till och med 200. Med strukturdiagram och pseudokod kan det se ut såhär:



**Figur 5.1:** Iteration som strukturdiagram.

```

1 start
2   tal = 1
3   så länge tal <= 200
    
```

```

4      skriv ut tal
5      tal = tal + 1
6  slut

```

Diamantstrukturen i strukturiagram används när det finns fler än en väg att gå. **F** står för falskt och **S** för sant. Vilket väg som skall tas beror på om villkoret inom diamanten är uppfyllt (S) eller ej (F). Notera att samma väg kan vandra flera gånger, det är själva poängen med iteration. Notera att värdet på variabeln `tal` ändras inom sant-banan. Om inte så hade vi fastnat i en oändlig loop. När villkoret inte längre uppfylls vandrar man F-vägen och hamnar vid algoritmens slut.

Pseudokoden beskriver samma sak men notera indragen. Raderna `skriv ut tal` och `tal = tal + 1` har ett indrag mer än raden ovanför, vilket indikerar att dessa ”hör till” kodblocket `så länge tal <= 200`. Precis som hela algoritmen ”hör till” `start`-blocket. Det är alltså endast kodraderna med två indrag som kommer att upprepas.

När raden `tal = tal + 1` har körts återvänder man till att kontrollera villkoret `tal <= 200` en gång till, men nu med de nya förutsättningarna (variabeln `tal` har ju fått nytt värde). Detta synliggörs tydligare i strukturiagrammet än pseudokoden men principen är exakt densamma.

## 5.1 Iteration i C#

När vi skall skapa en iteration i C# så finns två olika sätt - **for**-sats och **while**-sats. Vi tittar på **for**-satsen först.

### 5.1.1 For-satsen

Om vi utgår från pseudokoden till exemplet med tal upp till 200 ovan så skriver vi i C# följande:

```

1  int tal;
2  for (tal = 1; tal <= 200; tal = tal + 1)
3  {
4      Console.WriteLine(tal);
5  }

```

Först deklarerar variabeln `tal` (kallas loopvariabeln). Sedan följer nyckelordet **for** samt en startparentes. På första ”platsen” i parentesen (tänk att ; separerar tre kodstycken inom parentesen) tilldelas `tal` värdet 1. Därefter kommer villkoret som ska uppfyllas för att köra loopen en gång till (`tal <= 200`) och till sist den förändring som skall ske efter varje loopvarv (`tal = tal + 1`). Med andra ord så är förändringen av variabelns värde ”inbyggt” i **for**-satsen syntax (skrivregler) och resultatet är då att i C#

är det endast utskriftsraden som är indragen (indenterad) till skillnad från pseudokoden. Förändringen hör dock till blocket på precis samma sätt men skrivs alltså något annorlunda.

Lägg märke till tecknen { och } som anger var ett block startar och slutar - det räcker alltså inte bara med indrag i C#! Indragen gör det mer läsligt för oss människor men dessa *curly brackets* (måsvingar) måste sättas ut för att kompilatorn ska kunna tolka koden korrekt.

En variant på C#-koden ovan hade varit att låta deklaration och tilldelning ske i ett enda steg - initialisering direkt i **for**-satsen.

```
1 for (int tal = 1; tal <= 200; tal = tal + 1)
2 {
3     Console.WriteLine(tal);
4 }
```

Ytterligare en förkortning av koden skulle kunna göras. Eftersom att det är väldigt vanligt att man ökar en variabls värde med ett så har just denna förändring fått en egen förkortning i C# (och många andra språk!) nämligen `tal++` (negativa motsvarigheten existerar givetvis också `--`). Med andra ord är `tal = tal + 1` exakt samma sak som `tal++`, inget likhetstecken behövs. Vår uppräkningskod skulle med andra ord då kunna se ut såhär:

```
1 for (int tal = 1; tal <= 200; tal++)
2 {
3     Console.WriteLine(tal);
4 }
```

Rent generellt kan vi beskriva **for**-satsen i C# enligt följande:

```
1 for (initialisering; villkor; förändring)
2 {
3     kod att upprepa
4 }
```

Några saker att lägga på minnet vad gäller **for**-satsen:

- Loopvariabeln måste inte vara ett heltal.
- Loopvariabelns förändring måste varken vara positiv eller i heltalssteg.
- Alla variabler som deklarerats före for-loopen kan användas (och tilldelas nya värden) inuti for-loop-blocket.
- Alla variabler som deklarerats inuti for-loopen kan **endast** användas inuti for-loop-blocket.
- Det är väldigt vanligt att variabelnamnet på loopvariabeln är `i`.
- I Visual Studio får du enkelt hjälp med att skriva en for-loop genom att endast skriva **for** och sedan trycka tab två gånger. Tryck sedan tab igen för att redigera de standardvärden som dyker

upp och avsluta sedan med enter för att ställa dig inuti loop-blocket.

### 5.1.2 While-satsen

Ett annat sätt att skapa iteration i C# är att använda **while**-satsen. Vi åstadkommer samma resultat som med **for**-satsen (och de beskrivs därmed med samma strukturdiagram och pseudokod) men skrivsättet är något annorlunda. Vi tar exemplet med tal upp till 200 en gång till:

```
1 int tal = 1;
2 while (tal <= 200)
3 {
4     Console.WriteLine(tal);
5     tal = tal + 1;
6 }
```

Som du märker så är sättet att skriva en **while**-sats något mer likt sättet vi beskriver iteration med pseudokod. Initialisering av loopvariabeln först, sedan upprepningvillkoret och slutligen *både* utskrift och förändring av variabelvärde inom loop-blocket. Precis som med **for**-sats kan vi skriva förändringen med förkortningen ++.

```
1 int tal = 1;
2 while (tal <= 200)
3 {
4     Console.WriteLine(tal);
5     tal++;
6 }
```

Några saker att lägga på minnet vad gäller **while**-satsen:

- Loopvariabeln måste inte vara ett heltal.
- Loopvariabelns förändring måste varken vara positiv eller i heltalssteg.
- Alla variabler som deklarerats före while-loopen kan användas (och tilldelas nya värden) inuti while-loop-blocket.
- Alla variabler som deklarerats inuti while-loopen kan **endast** användas inuti while-loop-blocket.

Som sagt kan både **for**- och **while**-loop används för att lösa samma problem men oftast brukar man säga att **for**-loopen är bäst lämpad när man vet antalet upprepningar som kommer att ske och att **while**-loopen är att föredra när antalet upprepningar är okänt (tills dess att programmet faktiskt kör igenom, såklart).

### 5.1.3 Enkla jämförelser

Hittills har vi uteslutande använt exemplet då vi jämför `tal` med 200 och det får vara antingen mindre än eller lika med. De fyra vanliga olikhetstecknen skrivs enligt följande i C#.

Vardagligt språk	Matematik	Källkod
Större än	$>$	<code>&gt;</code>
Mindre än	$<$	<code>&lt;</code>
Större än eller lika med	$\geq$	<code>&gt;=</code>
Mindre än eller lika med	$\leq$	<code>&lt;=</code>

## 5.2 Att göra

- 📖 Läs mer om iteration i boken s. 122-131.
- 📖 Läs mer om kodblock och programstruktur på s. 11 i boken.
- 🌐 Några ytterligare rader om for-loopar finns på [csharpskolan.se/article/for-satsen](http://csharpskolan.se/article/for-satsen).
- 🌐 Några ytterligare rader om while-loopar finns på [csharpskolan.se/article/while-satsen](http://csharpskolan.se/article/while-satsen).
- ✍️ Genomför övningarna 14-25 här nedan.
- ✅ Genomför självtestet *Iteration* på Moodle: [moodle.teed.se](http://moodle.teed.se).

## 5.3 Övningar

Lös följande uppgifter (skriv algoritmer som löser problemen) i tre steg: strukturdiagram, pseudokod och sedan källkod i C#.

14. Skriv ut talen 0-100.
15. Skriv ut talen 10, 9, 8, ... 2 och 1.
16. Läs in en multipel och räkna upp de första tio multiplerna. Exempel: Om värdet är 3 skall utskriften vara: 3 6 9 12 15 18 21 24 27 30.
17. Räkna från noll till ett okänt positivt tal som läses in.
18. För att skriva ut en multiplikationstabell används nedanstående algoritm. Beskriv den med ett strukturdiagram och provkör den sedan i C#.

```
1 start
```

```

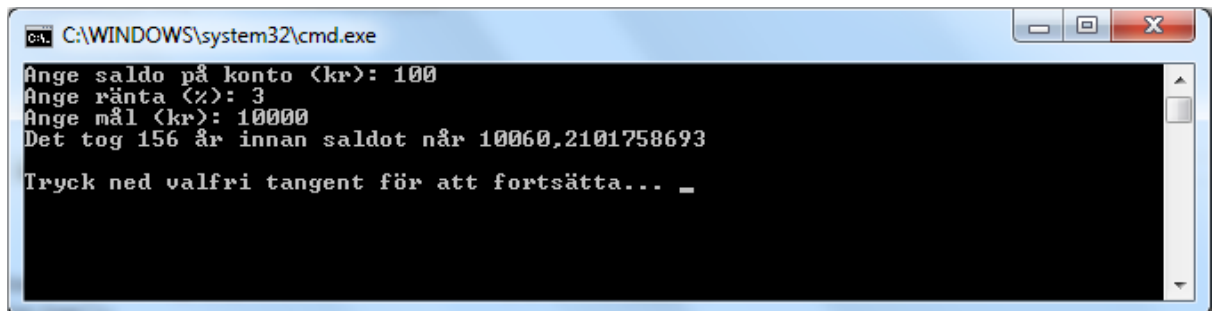
2   läs in tal
3   antal = 1
4   så länge antal < 11
5       skriv ut antal * tal
6       antal = antal + 1
7   slut

```

19. Räkna upp från okänt startvärde till okänt slutvärde.
20. Räkna ut summan av ett okänt antal positiva tal. Avsluta när summan är över en miljon.

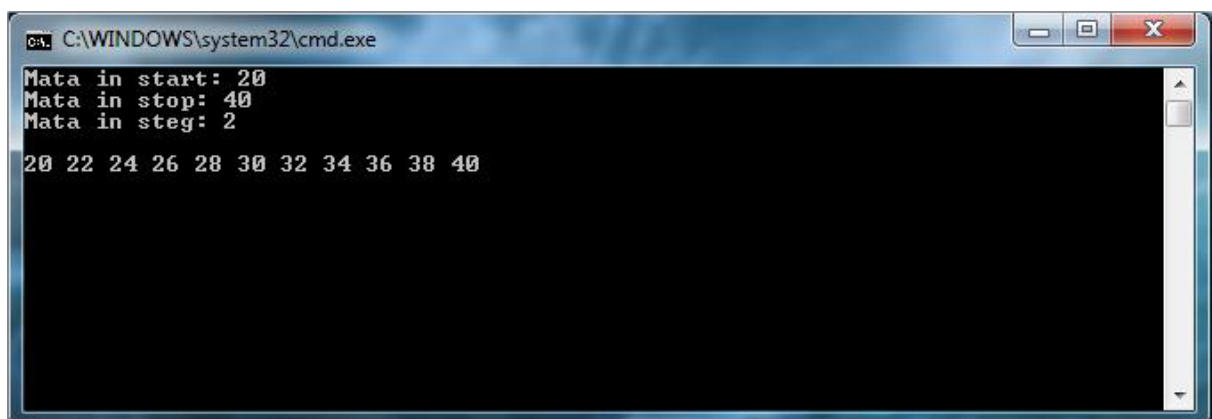
Kommande uppgifter behöver du endast lösa med källkod i C#.

21. Skapa en oändlig loop som skriver ut valfri text. Tips: Ctrl+c avbryter ett konsollprogram.
22. Skriv ett program där man får mata in 3 saker; saldo, ränta samt slutmål. Programmet ska sedan tala om hur många år det tar innan man når eller passerar slutmålet. Ange räntan i procentenheter.



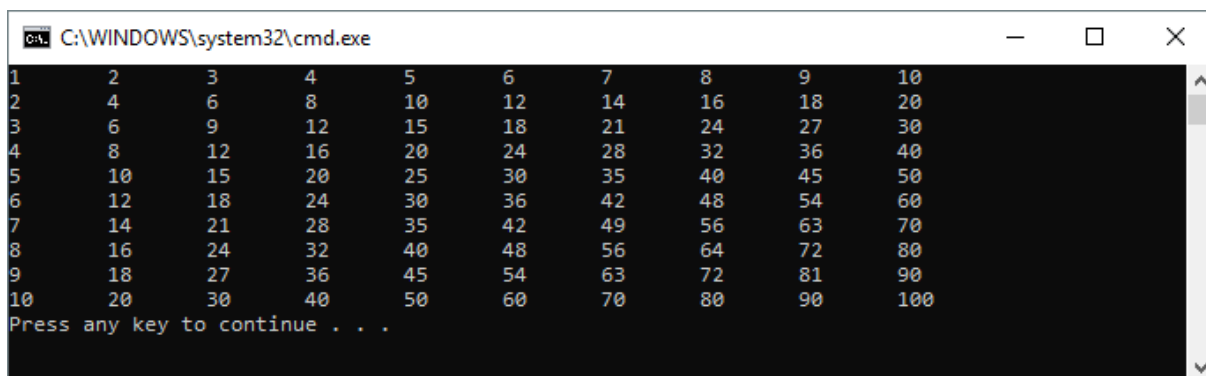
**Figur 5.2:** Testkörning av programmet med beräkning av ränta.

23. Konstruera ett program som ber om inmatning för "start", "stop" och "steg". Programmet skall sedan baserat på inmatningen göra en for-loop och skriva ut de tal som styr for-loopen enligt bild nedan.



**Figur 5.3:** Exempel på körning av programmet med steg för for-loop.

24. Skriv ett program som listar tal i Fibonacci-serien. Stanna när du passerat 1 miljon.
25. a) Skriv ett program som i "tabellformat" skriver ut multiplikationstabellerna med korrekt formatering enligt nedan (tips: nästlade loopar).

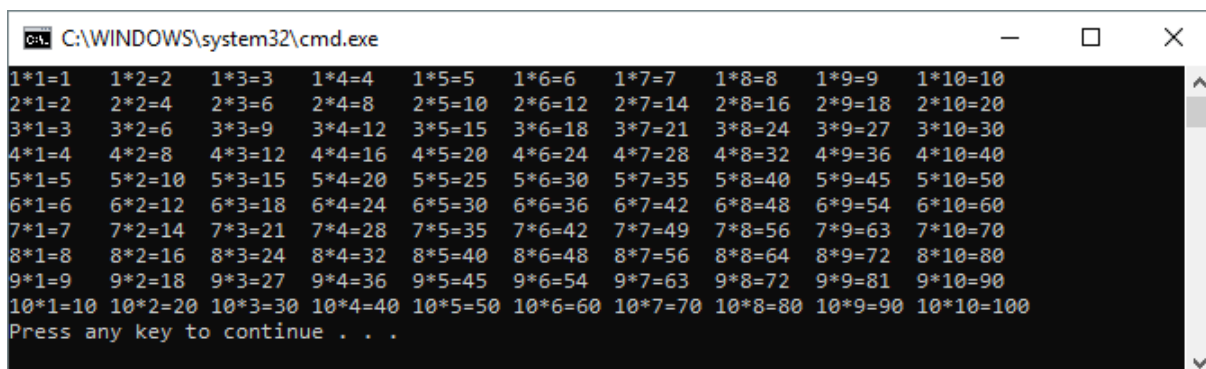


```

C:\WINDOWS\system32\cmd.exe
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
Press any key to continue . . .
  
```

**Figur 5.4:** Multiplikationstabellerna.

25. b) Utöka ditt program med multiplikationstabellerna så att även faktorerna skrivs ut.



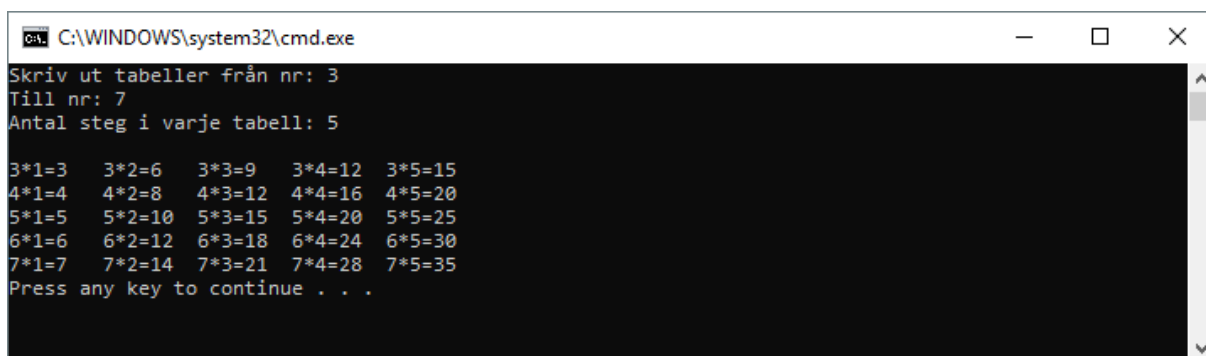
```

C:\WINDOWS\system32\cmd.exe
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 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27 3*10=30
4*1=4 4*2=8 4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36 4*10=40
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45 5*10=50
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54 6*10=60
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63 7*10=70
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72 8*10=80
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81 9*10=90
10*1=10 10*2=20 10*3=30 10*4=40 10*5=50 10*6=60 10*7=70 10*8=80 10*9=90 10*10=100
Press any key to continue . . .
  
```

**Figur 5.5:** Multiplikationstabellerna med faktorer.

25. c) Redigera i ditt program med multiplikationstabellerna så att man kan välja vilka tabeller som ska skrivas ut och hur många på varje rad.





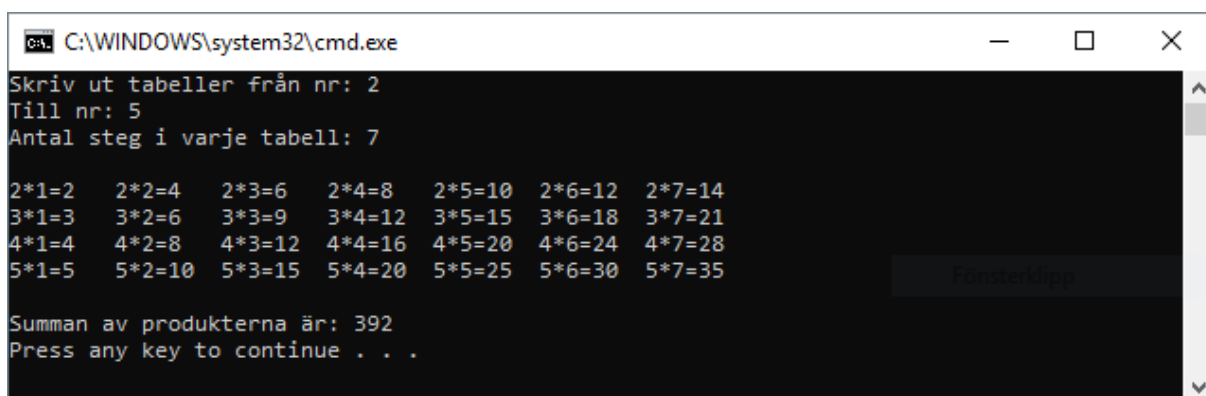
```

C:\WINDOWS\system32\cmd.exe
Skriv ut tabeller från nr: 3
Till nr: 7
Antal steg i varje tabell: 5

3*1=3  3*2=6  3*3=9  3*4=12  3*5=15
4*1=4  4*2=8  4*3=12  4*4=16  4*5=20
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25
6*1=6  6*2=12  6*3=18  6*4=24  6*5=30
7*1=7  7*2=14  7*3=21  7*4=28  7*5=35
Press any key to continue . . .
  
```

**Figur 5.6:** Multiplikationstabellerna med val.

25. d) Utöka ditt program med multiplikationstabellerna så att summan av produkterna skrivs ut nedanför tabellerna.



```

C:\WINDOWS\system32\cmd.exe
Skriv ut tabeller från nr: 2
Till nr: 5
Antal steg i varje tabell: 7

2*1=2  2*2=4  2*3=6  2*4=8  2*5=10  2*6=12  2*7=14
3*1=3  3*2=6  3*3=9  3*4=12  3*5=15  3*6=18  3*7=21
4*1=4  4*2=8  4*3=12  4*4=16  4*5=20  4*6=24  4*7=28
5*1=5  5*2=10  5*3=15  5*4=20  5*5=25  5*6=30  5*7=35

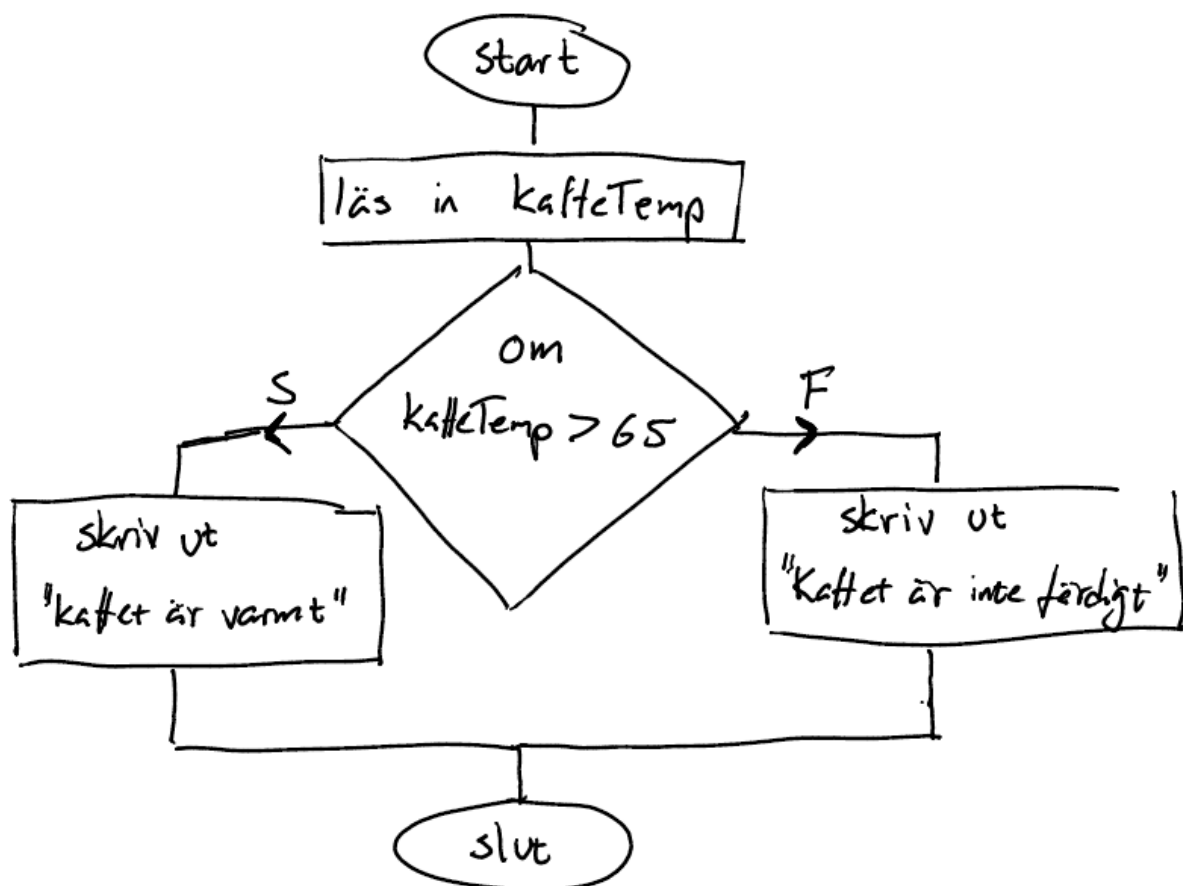
Summan av produkterna är: 392
Press any key to continue . . .
  
```

**Figur 5.7:** Multiplikationstabellerna med produktsumma.

## 6 Selektion

I förra avsnittet lärde vi oss hur en upprepning kan ske baserat på olika villkor. Nu kommer vi istället titta på hur man kan få programmet att göra en eller flera olika saker baserat på ett eller flera villkor. Med andra ord ska vi skapa olika "vägar" genom en algoritm. Vi kallar detta för selektion!

Om vi tänker oss ett exempel där beroende på inläst temperatur i en kopp med kaffe så skall texten "Kaffet är varmt!" eller "Kaffet är inte klart." skrivas ut. Gränsen för varmt kaffe sätter vi på 65 grader C. I form av strukturdiagram ser det då ut såhär:



Figur 6.1: Strukturdiagram med selektion.

Konstruktionen är densamma som för iteration (diamantstruktur) men texten inuti är annorlunda. Nu anger vi istället "om någonting" istället för "så länge som någonting". Båda vägar ut ur diamantstrukturen leder också vidare i programmet - inte tillbaka in i samma struktur som tidigare.

Om vi översätter ovanstående strukturdiagram till pseudokod skulle den se ut såhär:

```
1 start
2   läs in kaffetemp
3   om kaffetemp > 65
4       skriv ut "Kaffet är varmt!"
5   annars
6       skriv ut "Kaffet är inte klart."
7 slut
```

## 6.1 Selektion i C#

När vi sedan ska implementera detta (ovanstående algoritm) i källkod med C# så är nyckelordet vi använder **if**. Man säger ofta att man skriver en **if**-sats (eller om-sats, villkorssats, selektion). Direkt efter **if** sätter vi en parentes och inom parentesen det villkor som ska undersökas. Därefter kommer måsvingar **{}** som innehåller de kodstycken som ska köras om villkoret är sant. Efter måsvingarna kommer nyckelordet **else** följt av ett nytt par måsvingar. Inom dessa skriver man den kod som ska köras om villkoret är falskt. Såhär skulle det se ut:

```
1 Console.WriteLine("Mata in kaffetemp (C): ");
2 string inlasning = Console.ReadLine();
3 int kaffetemp = int.Parse(inlasning);
4 if (kaffetemp > 65)
5 {
6     Console.WriteLine("Kaffet är varmt!");
7 }
8 else
9 {
10    Console.WriteLine("Kaffet är inte klart.");
11 }
```

Det kan ju givetvis vara så att det finns flera olika villkor som ska undersökas. Vi skulle t.ex. kunna skriva ett program som avgör ifall ett inmatat tal är större än, mindre än eller lika med 100. Det blir då två villkor att undersöka. Först om det är större än 100, sen om det är mindre än 100. Om inget av dessa är sanna så måste det vara lika med 100. Källkoden skulle då kunna se ut såhär:

```

1 Console.WriteLine("Mata in ett tal: ");
2 string inlasning = Console.ReadLine();
3 int tal = int.Parse(inlasning);
4 if (tal > 100)
5 {
6     Console.WriteLine(tal + " är större än 100.");
7 }
8 else if (tal < 100)
9 {
10    Console.WriteLine(tal + " är mindre än 100.");
11 }
12 else
13 {
14    Console.WriteLine("Ditt tal är exakt 100.");
15 }

```

Lägg märke till raden med **else if**. Detta villkor kontrolleras endast om det ovanstående utvärderas till falskt. Dvs. om talet inte är större än hundra, kontrollera då om det är mindre än hundra. Utifall inget av dessa två villkor är sanna då körs kodstycket under **else** och endast då. Man har möjlighet att lägga till hur många **else if** som helst men så fort ett villkor som blir sant stöts på så hoppas resten av villkoren över - de kontrolleras inte ens.

Med villkorssatsen bör vi samtidigt lära oss några nya jämförelseoperatorer (vi har redan tittat på  $< > <= >=$ ). Vi vill kunna jämföra om något är exakt lika med (och endast lika med) eller skilt från (mer om operatorer generellt längre fram i kursen). Att jämföra likhet görs med dubbla likhetstecken **==** (det enkla likhetstecknet betyder ju tilldelning) och skilt från skrivs med utropstecken och sedan likhetstecken **!=**.

Vardagligt språk	Matematik	Källkod
Lika med	=	==
Skilt från	$\neq$	!=

Några exempel nedan.

Uttryck	Värde
2 == 3	falskt
2 != 3	sant
'a' != 'b'	sant

Uttryck	Värde
"Martin"== "Martin"	sant
5 + 7 == 12	sant
(3 * 3 == 12)== (15 < 12)	sant

Dessa uttryck kan göras väldigt långa och komplexa samt innehålla variabler men god praxis är att försöka hålla dom så läsbara som möjligt.

Eftersom att dessa villkor ska undersökas om de är sanna eller falska (inga andra utfall finns) så blir variabler av typen `bool` något mer användbara, särskilt i kombination med bra variabelnamn. Studera exemplet nedan:

```

1 Console.Write("Har du läst Matematik 4? (Ja/Nej): ");
2 string ma4 = Console.ReadLine();
3 Console.Write("Har du läst Fysik 2? (Ja/Nej): ");
4 string fy2 = Console.ReadLine();
5
6 bool kanBliIngenjor = false;
7 if (ma4 == "Ja")
8 {
9     if (fy2 == "Ja")
10    {
11        kanBliIngenjor = true;
12    }
13 }
14
15 if (kanBliIngenjor)
16 {
17     Console.WriteLine("Grattis, du kan bli ingenjör!");
18 }
19 else
20 {
21     Console.WriteLine("Du kan tyvärr inte bli ingenjör.");
22 }

```

Raden `if (kanBliIngenjor)` går ju att läsa upp på ren svenska/engelska - lättförståeligt för andra som läser våra program! Vi behöver inte jämföra om variabler av typen `bool` är lika med `true` eller `false` eftersom att det är de enda två värden som kan lagras och även de enda två värden som ett villkor kan utvärderas till - dvs. variabeln ÄR ju redan true eller false.

### 6.1.1 Flera villkor & logiska operatorer

Utöver jämförelseoperatorerna finns det också ett par logiska operatorer vi kan använda för att bygga mer komplexa villkorsuttryck.

Operator	Betydelse
!	Icke/negation
	Eller
&&	Och

- Icke (!) betyder helt enkelt motsatsen till uttrycket. Det som är sant görs falskt och det som är falskt görs sant.
- Eller (||) innebär att ett av två uttryck ska vara sant för att hela uttrycket ska bli sant. Antingen det ena *eller* det andra.
- Och (&&) medför att båda två uttryck måste vara sanna för att uttrycket som helhet ska bli sant. Det ena *och* det andra.

Vi tar några exempel:

Uttryck	Värde
<b>!true</b>	falskt
<b>!false</b>	sant
<b>true    false</b>	sant
<b>true &amp;&amp; false</b>	falskt
<b>5 == 1    -1 &lt; 0</b>	sant
<b>5 == 1 &amp;&amp; -1 &lt; 0</b>	falskt
<b>2 &lt; 3    5 &gt; 4</b>	sant
<b>2 &lt; 3 &amp;&amp; 5 &gt; 4</b>	sant

Med följande variabelvärden kan vi även titta på ytterligare exempel:

- **int** a = 7;
- **int** b = 10;
- **int** c = -3;

- `int d = 0;`
- `bool e = true;`

Uttryck	Värde
<code>a &gt; b</code>	falskt
<code>a != b</code>	sant
<code>b + c == a</code>	sant
<code>(a != c) &amp;&amp; (b + c != a)</code>	falskt
<code>e    (b &lt; a)</code>	sant
<code>!(c &gt; d)</code>	sant
<code>((a + b + c) &gt; d) &amp;&amp; !e</code>	falskt

## 6.2 Att göra

- 📖 Läs mer om selektion i boken s. 78-89.
- 🌐 Några ytterligare rader om if-satser finns på [csharpskolan.se/article/if-satsen](https://csharpskolan.se/article/if-satsen).
- ✍️ Genomför övningarna 26-39 här nedan.

## 6.3 Övningar

Genomför övningarna i tre steg: strukturdiagram, pseudokod och sedan källkod.

26. Läs in två tal och skriv ut vilket av dem som är störst.
27. Läs in två tal och skriv ut vilket av två tal som är minst. Om talen är lika skall detta skrivas ut.
28. Läs in mönsterdjupet på ett däck (mm). Om mönsterdjupen är större än 1,6 mm skall texten "Lagligt" skrivas ut, annars skrivs en text ut som berättar hur mycket som fattas för att det ska räknas som lagligt.
29. Lagra ett lösenord i en variabel. Låt användaren skriva in ett ord. Kontrollera om lösenordet är rätt eller fel.
30. Vad blir kostnaden för ett inköp om en affär ger 10 % rabatt på lösviktsgodis om vikten överstiger 5 hg? Läs in vikt i hg och priset per hg.

Nedanstående uppgifter behöver du endast lösa med källkod.

31. Läs in två tal och undersök om det första talet är jämnt delbart med det andra (tänk på att delat med-operatorn utför heltalsdivision).
32. Testkör de två kodstyckena nedan.

```

1  int tal = 11;
2  if (tal > 1)
3  {
4      Console.WriteLine("Större än 1.");
5  }
6  else if (tal > 10)
7  {
8      Console.WriteLine("Större än 10.");
9  }
10 else if (tal > 100)
11 {
12     Console.WriteLine("Större än 100.");
13 }

```

```

1  int tal = 11;
2  if (tal > 1)
3  {
4      Console.WriteLine("Större än 1.");
5  }
6  if (tal > 10)
7  {
8      Console.WriteLine("Större än 10.");
9  }
10 if (tal > 100)
11 {
12     Console.WriteLine("Större än 100.");
13 }

```

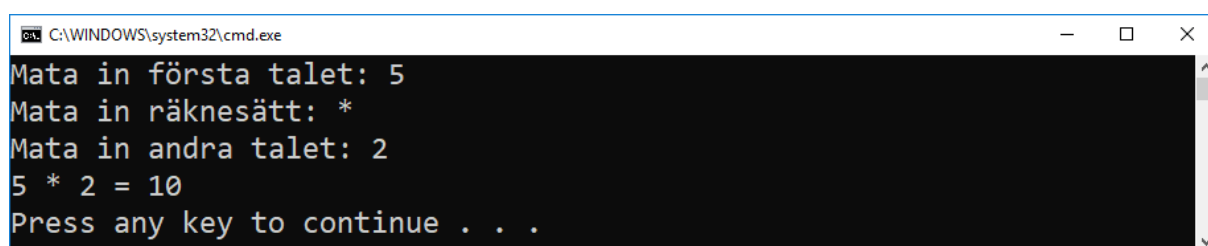
Vad skriver respektive program ut? Vad beror skillnaden på?

33. Skriv ut om en golfspelare klarar att undvika en vattenfylld damm. För att undvika dammen måste golfspelaren slå ett slag som är kortare än 150 meter eller längre än 190 meter.
34. Läs in ålder och skriv ut vad bussbiljetten kostar. Biljetten är gratis för barn under 4 år, från 4 till 17 år kostar den 16 kr. och om man är äldre än 17 kostar den 25 kr.
35. Du ska registrera ålder vid en mäsas. Beroende på ålder så ska besökarna få olika färger på sin bricka. Skriv ett program som svarar med rätt färg enligt tabellen:



Ålder	Färg
0-12	vit
13-18	grön
19-25	röd
26-99	blå
Övriga åldrar	Skriv ut "Ogiltig ålder"

36. Gör ett program som frågar efter ett tal. Efter inmatning undersöker programmet tecknet på talet och skriver ut om talet är positivt, 0 eller negativt.
37. Skriv ett program som fungerar som en enkel miniräknare. Användaren får mata in två tal samt ett räknesätt och programmet svarar med resultat. Om något annat tecken än + - \* / matas in på räknesätt så skrivs texten "Okänt räknesätt." ut.

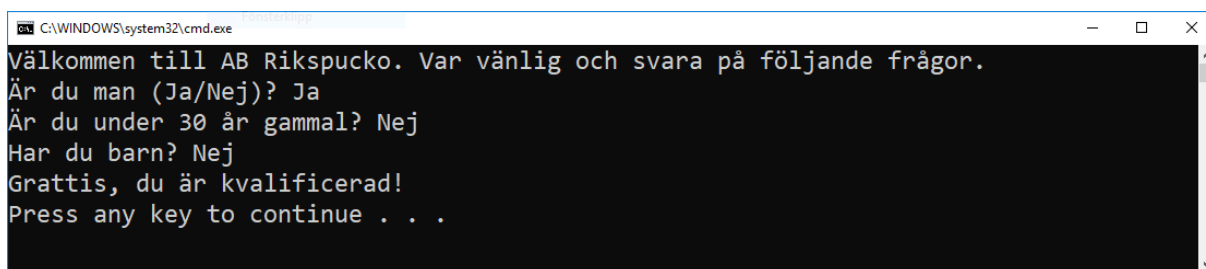


```

C:\WINDOWS\system32\cmd.exe
Mata in första talet: 5
Mata in räknesätt: *
Mata in andra talet: 2
5 * 2 = 10
Press any key to continue . . .
  
```

**Figur 6.2:** Exempelkörning av miniräknaren.

38. För att slå klubbrekord i en längdhoppstävling måste hoppet vara längre än 7,92 meter och vindstyrkan högst 2,0 m/s. Skriv ett program som låter användaren mata in värden och som skriver "Grattis, nytt rekord!" om det är ett nytt rekord. Om det inte är ett nytt rekord skriver programmet ut "Tyvärr, inget nytt rekord!". (Tips: Man kan skriva if-satser inuti if-satser)
39. Företaget AB Rikspucko söker nya anställda. För att få en tjänst måste två av tre villkor vara uppfyllda: vara man, vara yngre än 30 år och inte ha några barn. Skriv ett program som ställer tre frågor och som sedan visar om man är kvalificerad för en tjänst eller inte.



A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\WINDOWS\system32\cmd.exe' and the window name 'Fönsterklipp'. The command prompt displays the following text: 'Välkommen till AB Rikspucko. Var vänlig och svara på följande frågor.', 'Är du man (Ja/Nej)? Ja', 'Är du under 30 år gammal? Nej', 'Har du barn? Nej', 'Grattis, du är kvalificerad!', and 'Press any key to continue . . .'. The text is white on a black background.

**Figur 6.3:** Exempelkörning av AB Rikspucko.

## 7 Blandade övningar

Innan vi jobbar med blandade övningar (där sekvens, iteration och selektion blandas) tar vi en titt på några specialfall av iteration - nämligen villkorsoperatoren och switch-case-satsen.

### 7.1 Villkorsoperatoren

Villkorsoperatoren kan man säga är ett förkortat sätt att skriva en if-else-sats. Koden för villkoret och de två olika utfallen skall helst vara kort nog för att få plats på en rad för att det ska bli läsligt - men det är inget krav för att det ska fungera. Vi tittar först på en helt vanlig if-sats och sedan hur motsvarigheten skulle kunna skrivas med villkorsoperatoren:

```
1 int storst = 0;
2 if (13 > 37)
3 {
4     storst = 13;
5 }
6 else
7 {
8     storst = 37;
9 }
```

```
1 storst = 13 > 37 ? 13 : 37;
```

Vi använder villkorsoperatoren om det är en tilldelning som ska ske utifrån ett visst villkor. I det här fallet skall det största talet av 13 och 37 tilldelas till variabeln `storst`. I första exemplet har vi en if-sats i vanlig ordning. I andra exemplet gör vi exakt samma sak - men på endast *en* kodrad! Kod fungerar som så att direkt efter likhetstecknet (vanlig tilldelning, precis som med variabler tidigare) kommer det villkors som ska utvärderas följt av ett frågetecken. Precis efter frågetecknet skrivs det värde som variabeln `storst` ska få om uttrycket är sant. Detta följs av ett kolon (vanligt kolon, inte semikolon) och sedan det värde variabeln istället ska få om uttrycket är falskt. På "svenska" skulle man kunna säga att en kodrad med villkorsoperatoren kan läsas som `variabeln = uttryck ? värde om sant : värde om falskt;`.

Ett program som läser in två tal och skriver ut det största kan helt enkelt skrivas såhär:

```
1 Console.Write("Skriv in ett tal: ");
2 string inlasning = Console.ReadLine();
3 int tal1 = int.Parse(inlasning);
4
5 Console.Write("Skriv in ett andra tal: ");
6 inlasning = Console.ReadLine();
7 int tal2 = int.Parse(inlasning);
8
9 int storst = tal1 > tal2 ? tal1 : tal2;
10
11 Console.WriteLine(storst + " är störst.");
```

## 7.2 switch-case-satsen

Om det är ganska många olika villkors som skall kontrolleras (t.ex. som i uppgift 37 där vi gjorde en miniräknare, ett villkor för varje räkneoperation) så kan det bli lite stökigt/onödigt mycket kod för ganska lite funktionalitet. När det rör sig om att en och samma variabel kan anta flera olika värden kan det istället vara bra att använda *switch-case*. Man pekar ut vilken variabel man ska *switcha* på och sedan vilka olika *case* som finns. Exemplet med miniräknaren kan se ut såhär:

```
1 Console.Write("Skriv in ett tal: ");
2 string inlasning = Console.ReadLine();
3 int tal1 = int.Parse(inlasning);
4
5 Console.Write("Skriv in räknesätt: ");
6 inlasning = Console.ReadLine();
7 char raknesatt = char.Parse(inlasning);
8
9 Console.Write("Skriv in ett andra tal: ");
10 inlasning = Console.ReadLine();
11 int tal2 = int.Parse(inlasning);
12
13 switch (raknesatt)
14 {
15     case '+':
16         Console.WriteLine(tal1 + tal2);
17         break;
18     case '-':
19         Console.WriteLine(tal1 - tal2);
```

```

20         break;
21     case '*':
22         Console.WriteLine(tal1 * tal2);
23         break;
24     case '/':
25         Console.WriteLine(tal1 / tal2);
26         break;
27     default:
28         Console.WriteLine("Okänt räknesätt");
29         break;
30 }

```

Single-quote-tecknen (') är inget som är speciellt för switch-case - det är istället på grund av att vi hanterar en variabel av typen char. Hade det istället varit en string hade vi behövt skriva något citattecken ("), t.ex. `case "ja":` eller om det varit ett heltal hade vi inte behövt något specialtecken före eller efter: `case 13:`.

Nyckelordet **default** är motsvarigheten till **else** i vanlig if-sats. Det kodstycket körs om inget av villkoren ovan blir uppfyllt. **break** används för att satsen skall avbrytas när ett sant utfall funnits.

## 7.3 break

I switch-case-satsen använde vi **break**. Detta nyckelord går också att använda för att avbryta en loop - dvs. i **for**- eller **while**-sats. Låt oss säga att vi med en **for**-loop skall summera alla tal från 1 och uppåt men avbryta när vi kommit till en summa som är större än 22. Det är då möjligt att genom att kombinera **for**, **if** och **break** att ha en loop som löper enligt ett villkor men avbryts enligt ett annat. T.ex såhär:

```

1  int summa = 0;
2  for (int i = 1; i <= 100; i++)
3  {
4      summa = summa + i;
5      Console.WriteLine("Varv " + i + ": " + summa);
6      if (summa > 22)
7      {
8          break;
9      }
10 }
11 Console.WriteLine("Summan blev " + summa);

```

Trots att **break** sitter i en if-sats avbryts alltså den **senast påbörjade** loopen. Detta går även att använda i en while-sats, med andra ord kan du med hjälp av **break** avbryta en while-sats även om villkoret för att upprepa fortfarande är uppfyllt.

## 7.4 Slumptal

I några av de kommande uppgifterna är det nödvändigt att låta ditt program slumpa fram ett tal. Det gör vi enklast på följande vis.

```
1 Random slumpmaskin = new Random();
2 int slumptal = slumpmaskin.Next(1,11);
```

Koden ovan slumpar fram heltal mellan 1 och 10 (första siffran är inklusive och andra exklusive). Vill man slumpa tal mellan 0 och 100 anger man alltså (0, 101). Den resterande syntaxen är inget du behöver lära dig just nu (exempel finns även på bladet med syntaxhjälp) men man kan lära sig att första raden skapar en slumptionsgenerator och den andra raden slumpar fram ett heltal och lagrar i en variabel.

Samma generator (`slumpmaskin`) kan användas för att slumpa fram flera tal. Man kan alltså köra `slumpmaskin.Next(1,11)` flera gånger för att få fram flera slumptions. En ny slumptionsgenerator behöver ej (ska ej) skapas.

## 7.5 Kommentarer

Om man vill skriva saker i sin källkodsfil utan att detta påverkar programmets körning kan detta göras i form av kommentarer.

Kommentarer är tänkta som text till sig själv eller andra programmerare. De skrivs antingen på enkelrad med `//` eller på multirad med `/*` och `*/`. Det kan t.ex. se ut såhär:

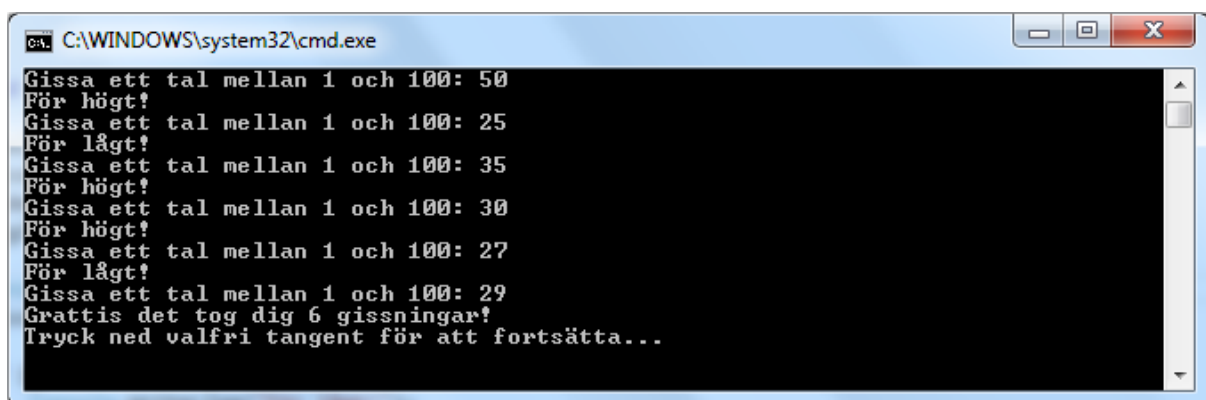
```
1 int totalpoang = 0; //Detta är en kommentar
2 double totalvarde = 0;
3 //Detta är en annan kommentar
4 for (int i = 1; i < 11; i++)
5 {
6     /*
7     Koden här nedan kommer inte köras eftersom
8     att den ligger inom en kommentar
9     totalpoang = totalpoang + 50;
10    */
```

11 }

Det är ofta en god idé att som programmerare lämna en kommentar i källkoden om det är något man programmerat som är lite mer svårförståeligt. Man förklarar kort det man gjort med en kommentar. Utgångspunkten är dock att man ska skriva så tydlig och läsbar kod (bra variabelnamn mm.) som möjligt att behovet av sådana kommentarer är minimalt. Välskriven kod behöver sällan kommenteras!

## 7.6 Övningar

40. Skapa ett "Gissa talet"-program. Programmet har ett hemligt tal som du ska gissa! Talet kan bestämmas på förväg i koden eller slumpas fram. Det hemliga talet ska ligga mellan 1 och 100. Användaren får gissa tills användaren gissat rätt. När användaren gissat rätt så ska antalet gissningar skrivas ut.



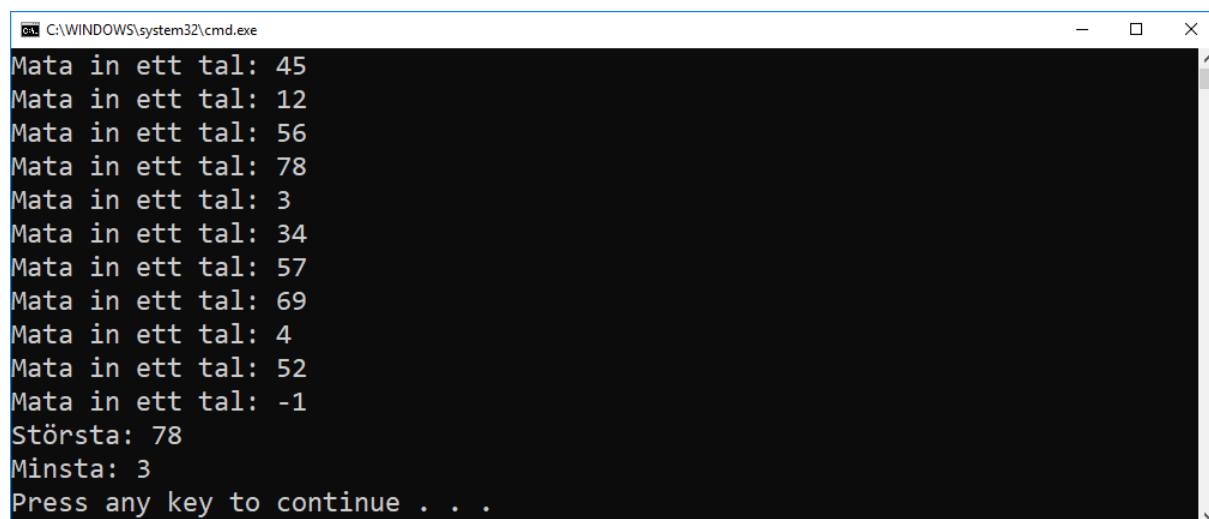
**Figur 7.1:** Gissa talet!

41. Skriv ett program som testat sig fram till en approximativ (läs ungefärlig) lösning till kubikroten ur ett tal. Använd en decimals noggrannhet (tips: använd en variabel av typen double för att styra for-loopen).
42. Skriv ett program där en användare har oändligt antal försök på sig att logga in med ett i källkod angivet användarnamn och lösenord. Så länge användaren skriver fel får denne fortsätta försöka logga in. Om inloggningen är korrekt skrivs ett välkomstmeddelande ut.
43. Gör ett program där användaren kan mata in numret på en veckodag. Programmet skall sedan skriva ut veckodagens namn. Använd en switch-sats i ditt program.
44. Gör ett program som frågar efter ditt kön och som ger svarsalternativen m och k. Därefter skall utskriften bli: "Du är en man" eller "Du är en kvinna", beroende på om du matat in ett 'm' eller inte. Använd villkorsoperatoren "?" i ditt program.

45. Skriv ett program som låter användaren mata in två decimaltal och sedan beräknar addition, subtraktion, multiplikation och division mellan talen. Programmet ska sedan skriva ut svaret av det räknesätt som ger störst resultat.
46. Vad innehåller strängen `a` när loopen är klar?

```
1 string a = "";
2 for (int i = 20; i >= 0; i = i - 5)
3 {
4     s = s + i;
5 }
```

47. Skriv ett program som läser in två tal och kollar om den ena är negativ och den andra positiv. I så fall skrivs "Sant" ut annars "Falskt".
48. Skapa ett program som skriver ut alla tal mellan 1 och 200 som är delbara med 3 och med 7.
49. Skapa ett liknande program som i uppgiften ovan men låt användaren bestämma vilka tal resultatet ska vara delbara med.
50. Skapa ett program där en användare kontinuerligt får mata in nya positiva heltal. Programmet ska sedan hålla reda på det största och det minsta talet som matats in. För att avsluta ska användaren mata in -1 och då skrivs största och minsta talen ut.



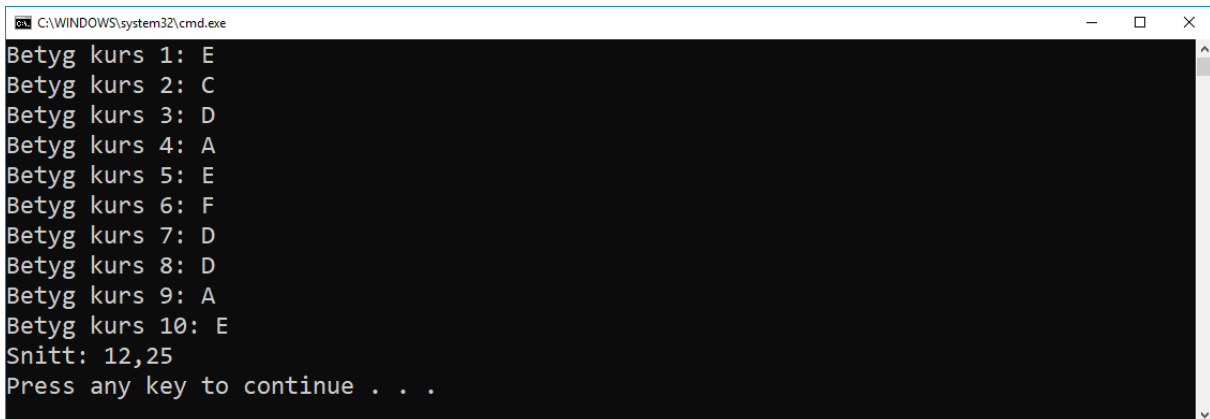
```
C:\WINDOWS\system32\cmd.exe
Mata in ett tal: 45
Mata in ett tal: 12
Mata in ett tal: 56
Mata in ett tal: 78
Mata in ett tal: 3
Mata in ett tal: 34
Mata in ett tal: 57
Mata in ett tal: 69
Mata in ett tal: 4
Mata in ett tal: 52
Mata in ett tal: -1
Största: 78
Minsta: 3
Press any key to continue . . .
```

**Figur 7.2:** Största och minsta vid flera inmatningar.

51. Skapa ett program som slumpar fram slumpstal mellan 1 och 6 (=simulera vanlig tärning). Räkna hur många 'kast' behövs för en 6:a.
52. Skapa ett program som genererar/kastar många slumpstal dvs flera tusen. Räkna hur många 6:or



- det blir och om det är 1/6 av alla kast.
53. Gör likadant som förra uppgiften men nu för alla tal.
  54. Skriv ett program som räknar hur många kast det behövs för att få en dublett, dvs två likadana valörer direkt efter varandra.
  55. Skriv ett program som räknar ut hur många kast som behövs innan alla tal har förekommit minst en gång.
  56. Skriv ett program där du får mata in betyg (A-F) på 10 gymnasiekurser i rad. Alla kurser är på 100p förutom den första som är på 50 och den sista som är på 150. Programmet ska sedan räkna ut snittbetyget och skriva ut på skärmen.



```

C:\WINDOWS\system32\cmd.exe
Betyg kurs 1: E
Betyg kurs 2: C
Betyg kurs 3: D
Betyg kurs 4: A
Betyg kurs 5: E
Betyg kurs 6: F
Betyg kurs 7: D
Betyg kurs 8: D
Betyg kurs 9: A
Betyg kurs 10: E
Snitt: 12,25
Press any key to continue . . .
  
```

**Figur 7.3:** Snittbetygsräknaren.