
Programmering 1

Del 2: Arrayer och metoder

Martin Larsson

19 juni 2019

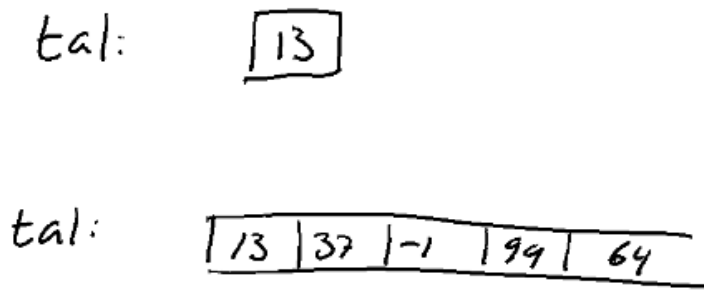
Innehåll

1	Arrayer	4
1.1	Gå igenom en array	6
1.2	Sträng som char-array	6
1.3	Unicode	7
1.4	Att göra	8
1.5	Övningar	8
2	Stränghantering	10
2.1	Strängmetoder	10
2.2	Formatera strängar	11
2.3	Att göra	12
2.4	Övningar	12
3	Datumhantering	14
3.1	Övningar	15
4	Math-klassen & fler operatorer	16
4.1	Konstanter	16
4.2	Matematiska funktioner	16
4.3	Fler operatorer	17
4.3.1	Modulus	17
4.3.2	Tilldelningsoperatorer	18
4.4	Att göra	18
4.4.1	Övningar	18
5	Flerdimensionella arrayer	20
5.1	Array av arrayer	21
5.2	Initialisera flerdimensionell array	23
5.3	Ytterligare dimensioner	23
5.4	Övningar	24

6	Metoder (funktioner)	26
6.1	Metoddefinition	26
6.2	Att anropa en metod	27
6.2.1	Lokala variabler	29
6.2.2	Kopior av värden	29
6.3	Returtyper	29
6.4	Array som parameter	30
6.5	Överlagrade metoder	31
6.6	Gemensamma variabler	32
6.7	Att göra	33
6.7.1	Övningar	33

1 Arrayer

När vi tidigare använt vanliga variabler så har vi kunnat lagra ett enda värde i respektive variabel. Ett heltal i en **int**-variabel, ett tecken i en **char**-variabel osv. Med hjälp av array (fält/vektor) kan vi lagra flera värden med ett och samma variabelnamn. Syntaxen för array är två hakparanteser `[]` och i exemplet heltal kan vi tänka oss att det ser ut ungefär såhär:

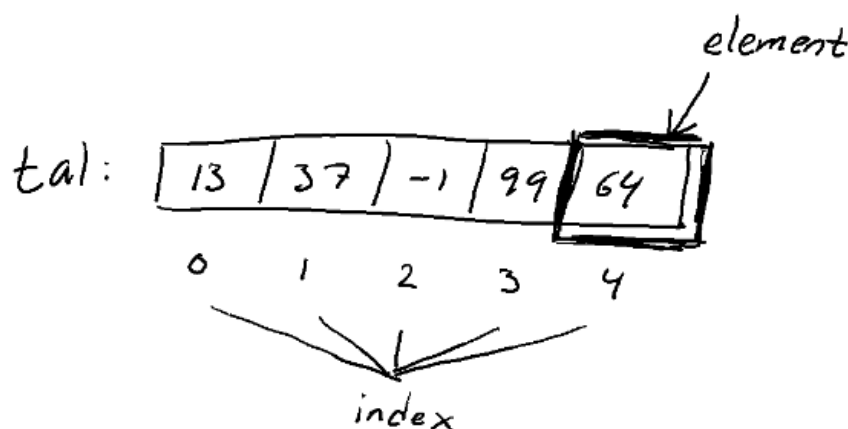


Figur 1.1: Vanlig heltalsvariabel och en heltalsarray.

Syntaxen för att skapa en array med några heltal ser ut som följande:

```
1 int[] tal = { 13, 37, -1, 99, 64 };
```

Varje tal hamnar på varsin plats och dessa platser numreras med start från 0. Denna numrering kallas index. Med andra ord finner vi det första talet på index 0, det andra talet på index 1 osv. Det som finns lagrat på respektive plats kallas element.



Figur 1.2: Delar av en array.

Utöver ovanstående sätt att skapa en array (med några tal specificerade från början) kan vi även skapa en tom array med ett antal platser. Det gör vi på följande vis:

```
1 int[] tal = new int[100];
```

Kodraden skapar en array med 100 platser, varje plats har ett null-värde (tomt värde) tills dess att något annat stoppas in på respektive plats. Det är alltså siffran 100 i den andra hakparantesen som anger storleken. Generellt är syntaxen för initialisering av en array likadan som tidigare initialiseringar: först datatypen `int[]`, sedan variabelnamnet `tal`, sedan likhetstecknet = och till sist det värde som ska tilldelas variabeln `new int[100]`.

Om vi ska tilldela nya värden till respektive plats i vår array använder vi arraynamnet i kombination med hakparanteser och index. Att stoppa in värden på de tre första platser kan man alltså göra såhär:

```
1 tal[0] = 13;
2 tal[1] = 37;
3 tal[2] = -1;
```

Vill vi istället skriva ut något, t.ex. från andra platsen refererar vi bara med arraynamnet, hakparanteserna och index 1.

```
1 Console.WriteLine(tal[1]);
```

Med andra ord fungerar varje plats i arrayen (`tal[i]`) precis som vilken variabel som helst. Det fina är dock att vi inte behöver skapa alla dessas variabler (100 st.) manuellt.

För att skapa arrayer av andra datatyper använder vi samma syntax, men istället för `int` skriver vi då givetvis den datatyp vi vill använda, t.ex:

```
1 double[] decimaltal = new double[10];
2 char[] tecken = new char[13];
3 string[] strangar = new string[80];
```

1.1 Gå igenom en array

På grund av att indexeringen av en array görs med heltal så är en for-loop med heltal som loop-variabel mycket användbar när en array skall gås igenom (t.ex. alla element skall skrivas ut). Vi använder även en egenskap som finns hos alla variabler som är arrayer, nämligen `Length`. `Length` ger oss antalet platser i en array (tänk på att största index är 1 mindre eftersom att vi räknar från 0, så kallade *off by one*-fel är väldigt vanliga).

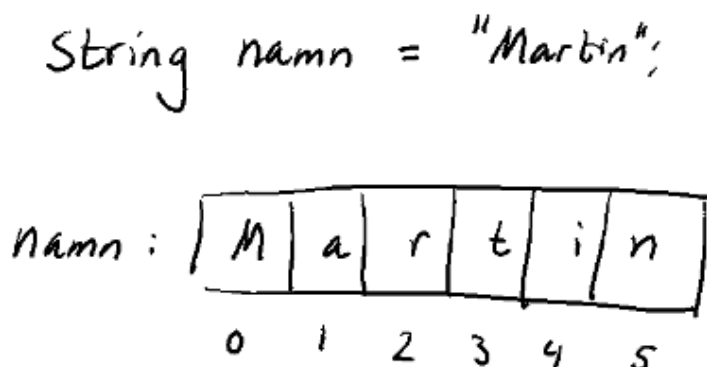
Vi tänker oss ett exempel där en användare ska få mata in 5 tal och dessa skall sedan skrivas ut i omvänd ordning.

```
1 int[] tal = new int[5];
2
3 for (int i = 0; i < tal.Length; i++)
4 {
5     tal[i] = int.Parse(Console.ReadLine());
6 }
7
8 for (int i = tal.Length - 1; i >= 0; i--)
9 {
10    Console.WriteLine(tal[i]);
11 }
```

I exemplet ovan bakar vi ihop `Console.ReadLine()` och `int.Parse()` utan mellanlagring i `string`-variabel för att göra programmet något kortare.

1.2 Sträng som char-array

Utan att veta om det har vi egentligen redan hanterat arrayer från första början. Det är nämligen så att alla strängar (`string`) egentligen är arrayer av tecken (`char`). Om jag t.ex. har texten `"Martin"` lagrad i en variabel `namn` består den egentligen av följande array:



Figur 1.3: Sträng som char-array.

När en sträng skapas skapas alltså en array vars storlek beror på antalet tecken i strängen. Varje bokstav, mellanslag eller andra specialtecken hamnar på varsin plats i arrayen. Om jag t.ex. ska skriva ut bokstäverna i mitt namn på varsin rad skulle jag kunna göra såhär:

```
1 string namn = "Martin";
2 for (int i = 0; i < namn.Length; i++)
3 {
4     Console.WriteLine(namn[i]);
5 }
```






Det krävs med andra ord ingen konvertering eller något annat kommando för att skapa arrayen i detta fall - att skapa strängen är att skapa char-arrayen!

1.3 Unicode

En annan sak som är något speciell med datatypen **char** är att varje tecken även har ett numeriskt värde. Om vi t.ex. initialiserar **char alpha = 'A'**; så skulle vi precis lika gärna kunna skriva **char alpha = (char)65**;. Tecknet A representeras alltså av värdet 65, datorns minne kan ju trots allt bara innehålla siffror. Koddelen (**char**) kallas för typkonvertering (eller *typecasting*) och innebär i detta fall en konvertering från **int** (65 står ju som ett vanligt heltal) till **char**. Ett alternativt sätt att konvertera är med kommandot **Convert.ToChar(65)**.

För att datorn ska veta vilket tecken som ska matchas till vilket tal används en stor teckentabell som i nuvarande form heter **Unicode**. Du hittar delar av denna på s. 39 i boken.

1.4 Att göra

-  Läs mer om arrayer i boken på s. 132.
-  Läs mer om tecken och Unicode i boken på s. 36-39.
-  Studera arrayer ytterligare på csharpskolan.se/article/falt-array.
-  Genomför självtestet *Arrayer* på Moodle: moodle.teed.se.
-  Genomför övningarna 1-10 här nedan.

1.5 Övningar

Genomför följande övningar i Visual Studio.

1. Skapa en heltalsarray med plats för fem tal. Stoppa in värdena 5, 3, 7, 12 och -3. Beräkna sedan summan för dessa
 1. Utan loop
 2. Med loop
2. Skriv ett program där du kan mata in ditt namn och programmet svarar med att skriva ut ditt namn baklänges.
3. Skapa en teckenarray med alfabetets 26 bokstäver (ej åäö).
 1. Skriv ut det tecken som finns på plats 17.
 2. Skriv ut index för bokstaven r.
4. Skapa en heltalsarray med 10 platser. Sätt in värdena -42, 13, 71, 0, 35, 8, 21, 99, -3 och 5.
 1. Beräkna summan av alla talen med hjälp av en loop.
 2. Skriv ut det största och det minsta talet.
 3. Beräkna produkten av alla tal som är större än 10.
5. Skapa ett program där storleken på en array bestäms av användarinput. Fyll arrayen med slumpade heltal inom ett intervall som användaren också bestämmer.
6. Skapa ett program där fem stycken tärningar slås (fem stycken slumpade tal mellan 1 och 6) och lagra antalet av varje valör i en array (antalet 1:or på index 0, antal 2:or index 1 osv...). Skapa sedan en kontroll för att undersöka om man har
 1. Par (Två av samma)
 2. Liten stege (1, 2, 3, 4, 5)
 3. Skriv ut informationen om par (och vilken valör), stege och/eller inget av dessa.

7. Skriv ett program där användaren först får mata in en sträng och sedan ett tecken. Programmet ska sedan svara med hur många av det inmatade tecknet det finns i strängen.
8. Skriv ett program som skapar en array med följande tal: -23, -20, -13, -84, -68, 79, -5, 71, 4, 58, -60, 46, -57, -44, -87, 62, -6, 99, 18, -64, 65, -72, 20, 8, -97, -28, 74, -59, -49, 39. Låt sedan användaren mata in ett tal mellan -99 och 99. Programmet ska svara med vilket index talet finns på om det existerar i arrayen, annars med texten "Finns ej."
9. Skriv ett program där du matar in ditt namn med versaler och programmet skrivet ut med gemener. Använd teckentabellen på s. 39 i boken och nyttja tipset som ges i Övning 3.8 på sidan precis innan tabellen.
10. På webbsidor numera är det vanligt att man har unika, läsbara adresser för respektive artikel/undersida. Ofta försöker man skapa en adress som liknar titeln för sidan (se bild nedan). Men mellanslag eller åä är ingen bra idé att ha med i webbadresser - därför behövs ibland en funktion som tar en vanliga titel (mening) och ändrar om en del. Skriv ett program som tar emot en mening men ersätter alla åä med a, ö med o och mellanslag med bindestreck. Gör dessutom alla tecken till gemener.



Figur 1.4: Adress och titel liknar varandra.

2 Stränghantering

Hittills har vi använt strängar vid inläsning och utskrift samt nu senast i array-delen visat på hur de egentligen består av en array med tecken.

I några av uppgifterna till array-delen fick du prova på att redigera befintliga strängar, t.ex. byta ut versaler mot gemener eller ersätta mellanslag med bindestreck. Detta är ett fullt fungerande sätt att jobba med strängar men är något omständigt. Eftersom att jobba med text på det sättet är ett vanligt förekommande problem inom programmering har de flesta programspråk inbyggda funktioner för att hantera strängar på olika vis. Så även C#.

2.1 Strängmetoder

Vi kommer här att titta på några av de vanligaste metoderna för att jobba med strängar på olika sätt. Beskrivningarna i tabellen nedan utgår från att vi har följande kod:

```
1 string text = "En apa och en katt, vilken underbar skatt! ";
2 text.
```

Precis efter punkten på andra raden fyller vi på med respektive metod. Man använder alltså metoderna med `variabelnamn.Metodnamn`.

Metod	Beskrivning
<code>Contains(värde)</code>	Undersöker om <i>värde</i> finns någonstans i strängen. Returnerar bool.
<code>EndsWith(värde)</code>	Undersöker om <i>värde</i> finns i slutet av strängen. Returnerar bool.
<code>IndexOf(värde)</code>	Returnerar ett heltal för det index där <i>värde</i> finns i strängen. Om det inte finns returneras -1. <i>värde</i> kan vara både en sträng eller en char.
<code>Insert(start, värde)</code>	Returnerar en ny sträng där <i>värde</i> är inskjutit på plats <i>start</i> .

Metod	Beskrivning
<code>Remove(start)</code>	Returnerar en ny sträng där alla bokstäver från värdet <i>start</i> till slutet av nuvarande sträng har plockats bort.
<code>Replace(gammal, ny)</code>	Returnerar en ny sträng där värdet i <i>gammal</i> har ersatts med värdet i <i>ny</i> . <i>gammal</i> och <i>ny</i> kan vara både sträng och char.
<code>Split(värde)</code>	Delar upp strängen i olika delar vid varje förekomst av <i>värde</i> . Returnerar en string-array.
<code>Substring(start, längd)</code>	Plockar ut en del av en sträng och returnerar som en ny sträng. Börjar vid index <i>start</i> och går <i>längd</i> steg framåt.
<code>StartsWith(värde)</code>	Undersöker om <i>värde</i> finns i början av strängen. Returnerar bool.
<code>ToLower()</code>	Returnerar en kopia av strängen med samtliga bokstäver som gemener.
<code>ToUpper()</code>	Returnerar en kopia av strängen med samtliga bokstäver som versaler.
<code>Trim()</code>	Returnerar en ny sträng där alla inledande och avslutande white-space-tecken (mellanslag, tabb) plockats bort.

2.2 Formatera strängar

Vi har tidigare lärt oss specialtecknen `\n` och `\t` för att göra radbrytningar respektive tabb. Vi har även tittat snabbt på ett alternativt sätt att skriva ut värdet av en variabel mitt i en sträng. Vi använde då platshållare. Det kan se ut såhär, t.ex:

```
1 int alder = 31;
2 string namn = "Martin";
3 Console.WriteLine("Mitt namn är {0} och jag är {1} år gammal.", namn,
    alder);
```

Det är också fullt möjligt att använda samma platshållare flera gånger:

```
1 string bla = "Blå";
2 Console.WriteLine("Röd, Grön, {0}, Röd, Grön, {0}", bla);
```



Vi kan utöka användningen av dessa platshållare för att t.ex. formatera decimaltal. I kombination med tabbar kan vi skapa en lite snyggare tabell.

```
1 Console.WriteLine("Decimaler:\t1\t2\t3\t4");
2 Console.WriteLine("\t\t{0:0.0}\t{0:0.00}\t{0:0.000}\t{0:0.0000}", Math.E);
```

```
1 Console.WriteLine("{0:000}\t{1:000}\t{2:000}\t{3:000}", 5, 13, 999, -1);
```

Antalet nollor efter kolon (:) men före punkt (.) anger hur många siffror heltal ska skrivas ut med. Nollorna efter punkten anger antalet decimaler.

2.3 Att göra

-  Läs mer om stränghantering på csharpskolan.se/article/stranghantering/.
-  Genomför övningarna 11-19 här nedan.

2.4 Övningar

Genomför nedanstående övningar i Visual Studio.

11. Skriv ett program som låter dig mata in ditt namn och svarar med dina initialer. Så "Erik Dahlberg" blir "ED".
12. Genomför övning 10 igen men använd nu istället sträng-metoderna ovan.
13. Låt användaren mata in en text och låt programmet avgöra om texten går att använda som lösenord om följande krav ska uppfyllas:
 - Får ej innehålla mellanslag
 - Måste innehålla minst en stor bokstav
 - Måste innehålla minst en liten bokstav
 - Måste bestå av minst 8 tecken.
14. Skriv ett program som undersöker om en inmatad sträng är en godkänd mailadress.
 - Finns @ med
 - Finns något före @
 - Finns något efter @
 - Har delen efter @ något på formen domän.förkortning, t.ex. hotmail.com eller jonkoping.se.
15. Skriv ett program som undersöker om den text användaren matar in är ett palindrom. Bortse från mellanslag!

16. Skriv ett program som räknar antal ord i en sträng som användaren får mata in.
17. Bekanta dig med metodhjälpen som ges av Visual Studio. Skapa en sträng `string namn = "Jerry Seinfeld"`. Skriv sedan på nästa rad `namn`. och se den lista av funktioner som dyker upp. Många av dem bör du känna igen från tabellen ovan i detta häfte. Skriv ut en av funktioner, t.ex. Substring följt av tomma paranteser (). Du bör då få upp nästa "hjälpruta" med information om vad som ska stå i parantesen. Du kan även bläddra uppåt/nedåt med piltangenterna för att se olika varianter av samma funktioner, dvs. samma funktionsnamn men olika så kallade parametrar (mer om detta längre fram i häftet). Lägg detta på minnen och använd framöver vid behov.
18. Skriv ett program som räknar antalet gånger en delsträng finns inuti en sträng. (T.ex: I strängen "Martin Larsson" finns delsträngen "ar" två gånger.)
19. Skriv ett program som läser in en rad på formen `förnamn efternamn, ålder, stad, yrke` och sedan skriver ut information på formen

```
1 efternamn, förnamn
2 ---
3 Ålder: ålder
4 Stad: stad
5 Yrke: yrke
```

3 Datumhantering

`DateTime`-objektet i C# är mycket användbart om man ska arbeta med datum och/eller tid. Istället för att själv göra omräkningar mellan år, månader, dagar, timmar, sekunder etc. finns flertalet färdiga metoder som sköter detta åt oss och dessutom ger vår kod relativt god läsbarhet. Lägg så många som möjligt av nedanstående metoder och egenskaper på minnet.

Metod/Egenskap	Returnerar/Används till
<code>new DateTime(2018, 8, 13)</code>	Skapar ett nytt <code>DateTime</code> -objekt med datumet 13 augusti 2018.
<code>new DateTime(2018, 8, 13, 9, 0, 0)</code>	Skapar ett nytt <code>DateTime</code> -objekt med datumet 13 augusti 2018 och tiden 09:00:00.
<code>DateTime.Now</code>	Skapar ett nytt <code>DateTime</code> -objekt med dagens datum och nuvarande tid.
<code>DateTime.Today</code>	Skapar ett nytt <code>DateTime</code> -objekt med dagens datum.
<code>DateTime.DaysInMonth(2018, 2)</code>	Returnera antalet dagar i februari 2018.
<code>DateTime.IsLeapYear(2019)</code>	Returnerar sant/falskt om 2019 är skottår eller inte.

Om ett `DateTime`-objekt har skapats (`DateTime date = new DateTime();`) och tilldelats till en variabel kan sedan (bland annat) följande metoder/egenskaper användas.

Metod/Egenskap	Returnerar/Används till
<code>date.AddDays(3)</code>	Returnerar ett nytt <code>DateTime</code> -objekt med ett datum tre dagar framåt jämfört med <code>date</code> .
<code>date.AddSeconds()</code> , <code>date.AddMinutes()</code> , <code>date.AddHours()</code> ,	<code>date.AddMonths()</code> , <code>date.AddYears()</code>
<code>date.DayOfWeek</code>	Returnerar veckodag för <code>date</code> .

Metod/Egenskap	Returnerar/Används till
<code>date.DayOfYear</code>	Returnerar dag på året för <code>date</code> (t.ex. är 2018-10-23 dag 296)
<code>date.Year</code> , <code>date.Month</code> , <code>date.Day</code> , <code>date.Hour</code> , <code>date.Minute</code> , <code>date.Second</code>	Returnerar respektive enhet för <code>date</code> .
<code>date.ToShortDateString()</code>	Returnerar en sträng med datumet för <code>date</code> på formen "yyyy-mm-dd".

Tänk på att många av ovanstående behöver man använda `ToString()` på vid jämförelse med t.ex. en viss veckodag.

3.1 Övningar

Besvara nedanstående frågor genom att skriva små program i Visual Studio där du använder `DateTime` och de tillhörande egenskaperna/metoderna.

20. Årets treor på ED tar studenten 2019-06-14. Vilken veckodag och dag på året är detta?
21. Vad kommer det vara för veckodag på dina närmaste 10 födelsedagar?
22. Hur många skottår kommer du ha varit med om innan du fyllt 30?
23. Vilka datum är dom närmsta fem kommande fredag den 13:e?
24. Redigera i din lösning till övning 24 så att de fem senaste fredag den 13:e visas istället.
25. Hur många dagar i februari månad har passerat sedan 2000-01-01?

4 Math-klassen & fler operatorer

Precis som med `DateTime` och strängar finns en hel del användbara funktioner vad gäller matematik färdiga att använda i C#. Den klass vi använder heter `Math` och där kommer vi åt både matematiska funktioner och konstanter.

4.1 Konstanter

`Math`-klassen har två konstanter - naturliga talet e och π . Du kommer åt dem med `Math.E` respektive `Math.PI` och du behöver ej skapa ett nytt objekt av typen `Math` (faktum är att du inte *kan* skapa ett objekt av typen `Math`, med andra ord: `Math m = new Math();` ger ett kompileringsfel).

De båda konstanterna skrivs ut med följande decimalprecision:

- $\pi = 3,14159265358979$
- $e = 2,71828182845905$

4.2 Matematiska funktioner

Precis som med konstanterna kan du direkt använda de matematiska funktionerna genom att skriva `Math.Funktionsnamn` (eftersom att du inte ens kan skapa en `new Math()`). Värdet som respektive funktion räknar fram *returneras* av funktionen, vilket betyder att man bör lagra det i en variabel (om det ska användas senare i programmet). Du bör alltså skriva något i stil med:

```
1 double d = Math.Round(2.67, 1);
```

Utöver funktionerna i nedanstående tabell finns även samtliga trigonometriska funktioner.

Funktion	Användning/beskrivning
<code>Math.Round(decimaltal, antalDecimaler)</code>	Avrundar talet <i>decimaltal</i> till antalet decimaler angivna med <i>antalDecimaler</i> .

Funktion	Användning/beskrivning
<code>Math.Ceiling(decimaltal)</code>	Tvingar avrundning av <i>decimaltal</i> uppåt till närmsta heltal.
<code>Math.Floor(decimaltal)</code>	Tvingar avrundning av <i>decimaltal</i> nedåt till närmsta heltal.
<code>Math.Pow(bas, exponent)</code>	Beräknar potensen med <i>bas</i> och <i>exponent</i> . Båda talen kan vara heltal eller decimaltal.
<code>Math.Abs(tal)</code>	Beräknar absolutbeloppet av <i>tal</i> . Fungerar både för heltal och decimaltal.
<code>Math.Sqrt(tal)</code>	Beräknar kvadratroten ur <i>tal</i> . Fungerar både för heltal och decimaltal.
<code>Math.Exp(exponent)</code>	Beräknar potens med <i>e</i> som bas och <i>exponent</i> som exponent.

4.3 Fler operatörer

Utöver själva `Math`-klassen finns några ytterligare operatörer som underlättar vår programmering en hel del.

4.3.1 Modulus

Modulus är en rest-operator, dvs. den beräknar resten (istället för kvoten) vid division. Tecknet för modulus är `%` (procenttecknet) men har inget med hundradelar att göra. Modulus blir t.ex. väldigt användbart när vi ska undersöka delbarhet. Om *a* är delbart med *b* så ger `a % b` resultaten 0 (ingen rest).

```

1 Console.WriteLine(5 % 2);
2 // 1
3
4 Console.WriteLine(7 % 8);
5 // 7
6
7 Console.WriteLine(26 % 13);
8 // 0

```



4.3.2 Tilldelningsoperatorer

När vi gör beräkningar i programmeringen så gör vi väldigt ofta också tilldelningar till en och samma variabel samtidigt. Till exempel när vi beräknar summan på det här sättet:

```
1 int summa = 0;
2 for (int i = 0; i < 10; i++)
3 {
4     summa = summa + i;
5 }
```

Ett enklare sätt att skriva raden `summa = summa + i;` är `summa += i;`, dvs vi kombinerar tilldelnings- och additionsoperatorn. På så vis slipper vi skriva variabelnamnet `summa` flera gånger. Det betyder helt enkelt "addera på *i* till *summa*". Motsvarigheter finns till de andra aritmetiska operatorerna `-=`, `*=`, `/=` och `%=`.

4.4 Att göra

-  Genomför självtestet *Strängar, DateTime och Math* på Moodle: moodle.teed.se.
-  Genomför övningarna 26-32 här nedan.

4.4.1 Övningar

26. Loopa igenom följande array och beräkna summan av decimaltalen. Avrunda till 2 decimaler.

```
1 double[] tal = { -12.1, 6.387, Math.E, 1.2, 16.47, 3.67 };
```

27. Skriv ett program där användaren får mata in två tal, *a* och *b*. Programmet svarar sedan med ifall *a* är delbart med *b*.
28. *FizzBuzz* är en klassisk programmeringsuppgift som ofta förekommer i kurser och i böcker. Skapa ett program som skriver ut alla tal mellan 1 och 100 men
- för multiplar av 3 skriv *Fizz* istället för talet,
 - för multiplar av 5 skriv *Buzz* istället för talet och
 - för multiplar av både 3 och 5 skriv *FizzBuzz* istället för talet.
29. Beräkna differensen mellan **summan av kvadraterna** och **kvadraten av summan** för alla tal mellan 1 och 1000.

$$1^2 + 2^2 + 3^2 + \dots 1000^2$$

$$(1 + 2 + 3 + \dots + 1000)^2$$

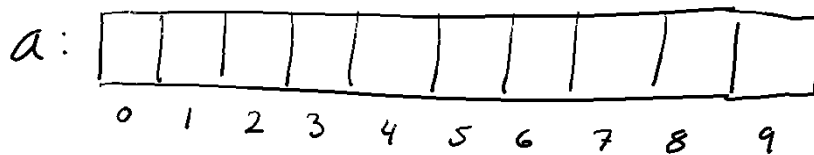
30. Skriv ett program (utan att använda `DateTime`) där användaren får mata in ett klockslag i form av två tal, timmar (0-23) och minuter (0-59). Därefter får användaren mata in ett tredje tal (i minuter med obegränsad storlek) som ska adderas till nuvarande tid. Programmet ska sedan visa det nya klockslaget korrekt på formatet hh:mm.
31. Skriv ett program som låter användaren ange värden för p och q i ekvationen $x^2 + px + q = 0$ och sedan ger två svar $x = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$
32. Skriv ett program som omvandlar ett inmatat tal i bas 10 till bas 2. Använd t.ex. matteguiden.se/matte-diskret/de-hela-talen/binara-och-hexadecimala-tal om du glömt av hur man gör.

5 Flerdimensionella arrayer

En array (ett fält) i en dimension har vi sett tidigare. Syntaxen ser ut såhär

```
1 int[] a = new int[10];
```

och vi kan visualisera arrayen på detta vis:

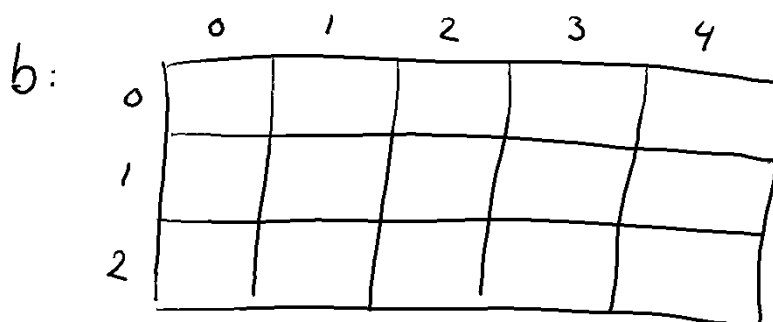


Figur 5.1: Endimensionell array av typen heltal.

Om vi istället går över till två dimensioner så kan vi använda liknande syntax men med några förändringar. En tvådimensionell array med storleken 5×3 (av typen heltal) skapar vi med

```
1 int[,] b = new int[5,3];
```

Ett kommatecken mellan de första hakparanteserna anger att det är fråga om två dimensioner. I den andra hakparantesen anger första siffran storleken på första dimensionen (bredden) och andra siffran storleken på andra dimensionen (höjden). Vi kan tänka oss den tvådimensionella arrayen som en matris.



Figur 5.2: Tvådimensionell array av typen heltal.

För att komma åt ett visst element (för att t.ex. läsa av eller ändra ett värde) måste vi ange två positioner (koordinater).

```
1 b[3,1] = 5;
```

En tvådimensionell array har alltså två siffror för sitt index. Båda räknar fortfarande från 0, dvs. i arrayen `b` som vi deklarerade ovan så finns index 0-4 i ena dimensionen och 0-2 i den andra.

Om storleken på en flerdimensionell array är okänd kan vi ta reda på den genom `GetLength()` (inte bara `Length`) och i parantesen ange ordningsnumret på den dimension man vill undersöka. Dessa nummer börjar också räknas från 0, precis som index. Om vi t.ex. ska loopa igenom matrisen `b` och stoppa in slumpade tal mellan 1 och 100 på varje position kan vi göra såhär:

```
1 int[, ] b = new int[5,3];
2 Random slump = new Random();
3 for (int i = 0; i < b.GetLength(0); i++)
4 {
5     for (int j = 0; j < b.GetLength(1); j++)
6     {
7         b[i,j] = slump.Next(1, 101);
8     }
9 }
```

5.1 Array av arrayer

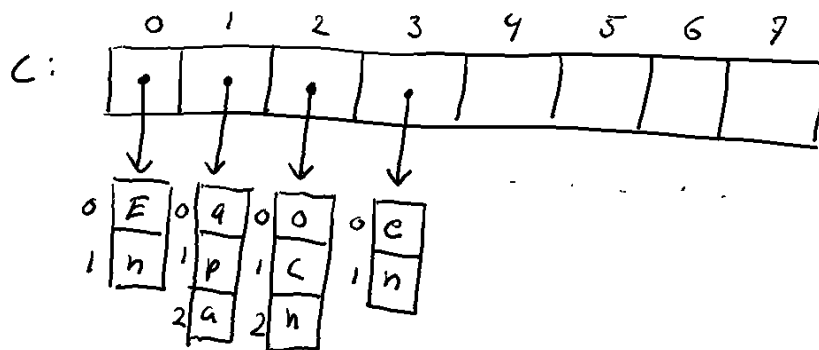
Det gäller att inte blanda ihop flerdimensionell array med *arrayer av arrayer*. Om vi t.ex. har en sträng `s`

```
1 string s = "En apa och en katt, vilken underbar skatt!";
```

och sedan kör `Split` på den

```
1 string[] c = s.Split();
```

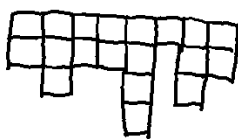
så kan vi **inte** komma åt första bokstaven i första ordet med `c[0,0]` utan måste ta till `c[0][0]`. Vi kan tänka oss att strängarrayen `c` (med dess char-arrayer) ser ut såhär:



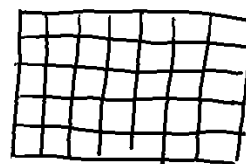
Figur 5.3: Array av arrayer.

Med andra ord gäller det att "den andra dimensionen" är olika stor för olika positioner i den första dimensionen, något som inte är sant för en matris.

Array av arrayer:



Matris:



Figur 5.4: Array av arrayer vs. matris

5.2 Initialisera flerdimensionell array

Om vi vill tilldela värdet till en flerdimensionell array direkt vid deklaration gäller det att sätta måsvingarna {} på rätt plats. Om en 3x3-matris ska fyllas med talen 1-9 kan det se ut såhär:

```
1 int[,] d =  
2 {  
3     {1,2,3},  
4     {4,5,6},  
5     {7,8,9}  
6 };
```

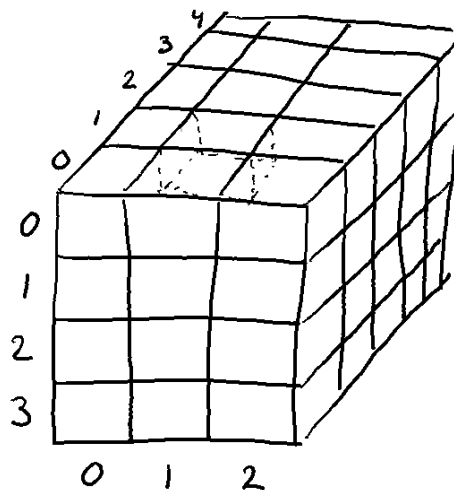
Med andra ord är det en upprepning av hur vi gör för att fylla en endimensionell array med ett par extra måsvingar som start och slut.

5.3 Ytterligare dimensioner

Om vi vill ha fler dimensioner är det bara att lägga till fler kommatecken i vår syntax! En tredimensionell array kan t.ex. skapas med

```
1 int[, ,] e = new int[3,4,5];
```

Och vi kan då tänka oss att strukturen ser ut ungefär såhär:



Figur 5.5: Array med tre dimensioner.

För att komma åt storleken (om den skulle vara okänd) på den tredje dimensionen får vi i så fall använda oss av `e.GetLength(2)`.

5.4 Övningar

33. Skapa följande matris. Summera sedan samtliga tal och skriv ut till skärmen.

```
1 int[,] tal =
2 {
3     { 3, 3, 9, 4, 3 },
4     { 5, 2, 1, 9, 8 },
5     { 6, 1, 8, 6, 7 },
6     { 5, 9, 5, 4, 8 },
7     { 4, 5, 1, 8, 8 }
8 };
```

34. Återanvänd datan ifrån uppgift 33 men beräkna nu istället summan kolumn för kolumn och skriv ut till skärmen. Med andra ord ska du skriva ut fem olika summor.
35. Skapa en 3x3-matris av typen `bool`. Sätt in värdet `true` på två godtyckliga ställen och `false` på resten. Låt sedan användaren ”spela” genom att skriva in koordinater och gissa var `true`-värdena finns. Skriv ut antal gissningar som krävdes och avsluta programmet när båda värdena har hittats.
36. I matrisen nedan betecknas levande cell med `*` och död cell med mellanslag. Skriv ett program som beräknar vilken cell som har flest levande grannar. En cell kan maximalt ha 8 grannar.

```
1 char[,] celler =
2 {
3     { ' ', '*', ' ', ' ', ' ', ' ', '*', ' ', ' ', ' ', '*' },
4     { ' ', '*', ' ', ' ', ' ', ' ', '*', ' ', ' ', ' ', '*' },
5     { '*', '*', '*', ' ', ' ', ' ', ' ', ' ', ' ', '*', ' ' },
6     { ' ', ' ', '*', '*', '*', '*', ' ', ' ', ' ', '*', ' ' },
7     { ' ', '*', ' ', ' ', ' ', ' ', '*', ' ', ' ', ' ', ' ' },
8     { ' ', '*', '*', ' ', ' ', '*', ' ', ' ', ' ', ' ', '*' },
9     { '*', '*', '*', ' ', ' ', ' ', ' ', '*', ' ', ' ', '*' },
10    { '*', '*', ' ', '*', ' ', ' ', ' ', ' ', '*', ' ', ' ' },
11    { ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '*', ' ' },
12    { ' ', '*', ' ', ' ', ' ', ' ', ' ', '*', '*', '*', ' ', ' ' },
13 };
```


37. Utöka ditt program från uppgift 36 och lägg till så att levande celler slumpas på exakt 30 celler. Få programmet att skriva ut matrisen i konsollen och sedan beräkna samt skriva ut hur många grannar cellen med flest grannar har (inklusive cellens koordinater).

6 Metoder (funktioner)

Hittills har vi skrivit all vår källkod direkt i det fördefinierade kodblocket `Main` men detta blir ganska snabbt lite stökigt, särskilt när våra program börja växa lite i antalet kodrader.

Ett sätt att dela upp programmet i mer hanterbara delar är att använda metoder (ibland benämns metoder också som funktioner, men utifrån hur programspråket C# är uppbyggt är det korrekta namnet egentligen metoder).

Du har använt massvis med metoder i dina program hittills, t.ex. är `Split` en metod som finns för strängar eller `Round` en metod som finns hos `Math`-klassen.

6.1 Metoddefinition

En metoddefinition har två delar - ett metodhuvud och en metodkropp. Metodhuvudet kallas också ibland för metodens signatur. Följande är ett exempel på en (våldigt enkel) metod:

```
1 int Addera(int a, int b)
2 {
3     int summa = a + b;
4     return summa;
5 }
```

På första raden ser vi metoden signatur och inom måsvingarna `{ }` ser vi metodens kropp. På signaturraden betyder de olika nyckelorden följande:

- **int**: Metodens returtyp. Det som metoden i slutändan svarar med skall vara av denna datatyp. I detta fall adderar metoden två heltal - alltså ska svaret vara ett heltal.
- `Addera`: Metodens namn. Detta namn bestämmer du som programmerare själv men ska vara så beskrivande som möjligt, precis som med variabelnamn. Metodnamn börjar alltid med stor bokstav. Består namnet av flera ord används samma princip som för variabelnamn (t.ex: `AdderaTvåTal`). Metodnamn är jämförbart med f i det matematiska uttryckssättet $f(x)$.
- **int** `a`: Datatyp och variabelnamn för den första parametern. Parameter är ett namn på den information som skickas med in i en metod. Jämför med x i $f(x)$.

- **int b**: Datatyp och variabelnamn för den andra parametern. Jämför med y i $f(x, y)$.

OBS! Antalet parametrar måste ej vara två, det kan vara 0 eller flera. Om en metod har 0 parametrar lämnas två tomma paranteser direkt efter namnet, annars separeras varje parameter med ett kommatecken.

För att få en metod att fungera korrekt i de konsollprogram vi skriver måste en sak till med i vår signatur - nämligen **static**. Eftersom att vi skriver all vår kod i metoden **Main** som i sig är **static** måste alla metoder vi använder i **Main** i sin tur vara **static** - en regel i C# som man helt enkelt får lära sig. Vi får då följande kod:

```
1 static int Addera(int a, int b)
2 {
3     int summa = a + b;
4     return summa;
5 }
```

Inuti metodens kropp kan du programmera helt enligt tidigare principer. Märk dock noga att så fort programmet stöter på nyckelordet **return** så kommer metoden att returnera (avslutas) och inte köra resten av koden som eventuellt kan finnas under. Första bästa **return** innebär alltså att metदानropet anses vara klart!

Överkurs: Det som **static** egentligen betyder är att det inte behövs ett objekt för att komma åt metoden, man kan arbeta direkt mot klassen. Jämför t.ex. **Math.Round()** med metoden vi använder för att få fram ett slumpstal **Next()**. För att komma åt **Next()** måste vi första skapa en **new Random()** men med **Round()** kan vi köra **Math.Round()** direkt utan att använda **new**. Mycket mer (och begripligare) om detta om du väljer att läsa kursen **Programmering 2** eller någon kurs i **objektorienterad programmering** på högskola/universitet.

6.2 Att anropa en metod

När metoden väl är skapad (huvudet + kroppen) skall vi givetvis använda den i vårt huvudprogram - annars är den ju inte till någon nytta. Vår källkod kan då se ut såhär:

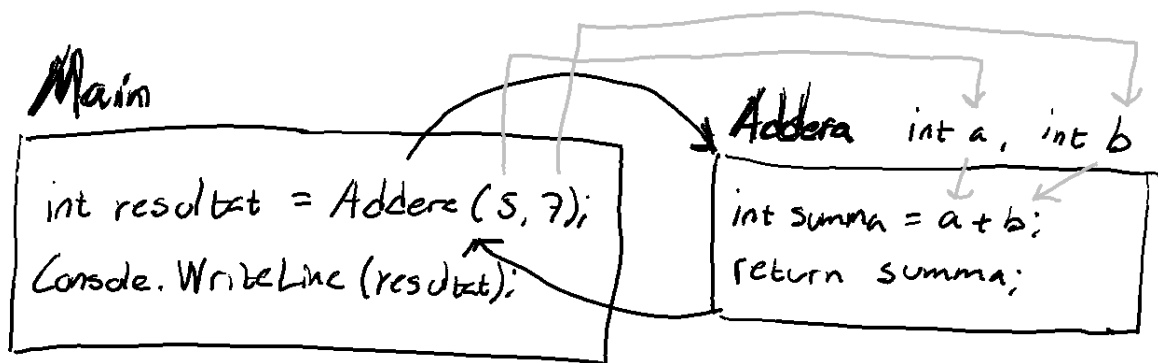
```
1 static void Main(string[] args)
2 {
3     int resultat = Addera(5, 7);
4     Console.WriteLine(resultat);
5 }
6
7 static int Addera(int a, int b)
```

```

8 {
9     int summa = a + b;
10    return summa;
11 }
  
```

Anropet till funktionen ser alltså ut såhär: `Addera(5, 7)`. Värdena 5 och 7 i detta fall kallas argument till metoden (det som skickas med). Svaret ifrån detta anrop (resultatet av koden i metoden) lagras i variabeln `resultat`. Eftersom att returtypen på `Addera` är `int` måste även datatypen på variabeln som tar emot resultatet vara `int`.

Det som händer vi körning är att när `Addera(5, 7)` anropas så får `a` värdet 5 och `b` värdet 7 i metoddefinitionen, beräkningen genomförs, svaret returneras och lagras sedan i variabeln `resultat`.



Figur 6.1: Ett metodanrop med return-sats.

Om samma metod anropas en gång till så har värdena på `a`, `b` och `summa` raderats. Det är en ny, färsk variant av metoden som körs andra gången.

```

1 static void Main(string[] args)
2 {
3     int resultat = Addera(5, 7);
4     Console.WriteLine(resultat); // 12
5     resultat = Addera(1, 2);
6     Console.WriteLine(resultat); // 3
7 }
8
9 static int Addera(int a, int b)
10 {
11     int summa = a + b;
12     return summa;
13 }
  
```

6.2.1 Lokala variabler

I exemplet ovan är `summa` ett exempel på en så kallad lokal variabel. Den existerar endast inuti metoden `Addera` och är ej åtkomlig utanför. När metoden har körts klart raderas värdet som lagrats i `summa` och minnet för variabeln frigjörs. Observera dock att värdet som returnerats (och lagrats i `resultat`) inte raderas. I tidigare nämnda exempel är 12 resultatet av att anropa `Addera(5, 7)`, den 12:an finns i huvudprogrammet och lagras i `resultat`. Metoden `Addera` har dock inget minne av varken 5, 7 eller 12 efter en genomkörning.

6.2.2 Kopior av värden

Parametrarna i en metods signatur får kopior av de värden som skickas med, det är med andra ord inte variabeln i sig som skickas iväg till en metod. Låt oss studera nedanstående exempel:

```

1  static void Main(string[] args)
2  {
3      int a = 5;
4      Console.WriteLine(a); // 5
5
6      int b = AdderaTre(a);
7      Console.WriteLine(a); // 5
8      Console.WriteLine(b); // 8
9  }
10
11 static int AdderaTre(int x)
12 {
13     x = x + 3;
14     return x;
15 }
```

Trots att `x` tilldelas nytt värde i `AdderaTre` så påverkas ej variabeln `a`, eftersom det är en kopia av `a` som skickas till metoden.

6.3 Returtyper

Alla datatyper vi jobbat med kan användas som returtyper (`int`, `double`, `bool`, `char`, `string` och alla arrayvarianter...). Vi lägger även till en ny till listan - `void` (tomrum på engelska). Denna returtyp

används om metoden inte ska returnera något alls, något som bland annat är vanligt om metoden t.ex. endast ska göra en utskrift av något. Något resultat skickas då alltså inte tillbaka till huvudprogrammet (eller där anropet skedde).

En metod som tar en **int**-array och endast gör en utskrift av denna kan se ut såhär:

```
1 static void SkrivUtArray(int[] tal)
2 {
3     for (int i = 0; i < tal.Length; i++)
4     {
5         Console.Write(tal[i] + " ");
6     }
7 }
```

När returtypen **void** används **ska** alltså inte nyckelordet **return** användas i metodkroppen. När en metod inte returnerar något kan inte heller resultatet av anropet lagras i en variabel (eftersom att det inte finns något resultat). Då ser anropet helt enkelt ut såhär:

```
1 int[] matningar = { 5, 7, -1, 32, 11, 17 };
2 SkrivUtArray(matningar);
```

6.4 Array som parameter

Ett specialfall vad gäller parametrar är arrayer. Eftersom att de som designat programspråket C# har haft (bland annat) prestanda i åtanke så blir en array-parameter **inte** en kopia som de andra parametrarna. Detta på grund av att en array mycket väl kan innehålla en stor datamängd och att göra en kopia på en sådan är väldigt resurskrävande. Istället blir arrayen i detta fall en så kallad referensparameter. Parameternamnet i metoden *pekar* på den "riktiga" variabeln i huvudprogrammet. Med andra ord blir en ändring av arrayens värden inuti metoden även en ändring i huvudprogrammet (även om vi inte returnerar något) - det är samma array! Koden nedan är exempel på detta:

```
1 static void Main(string[] args)
2 {
3     int[] minArray = { 1, 2, 3, 4, 5 };
4
5     // Före
6     for (int i = 0; i < minArray.Length; i++)
7     {
8         Console.Write(minArray[i] + " ");
9     }
```

```

10
11     Console.WriteLine();
12     MultipliceraMedFem(minArray); // Anropa metoden
13
14     // Efter
15     for (int i = 0; i < minArray.Length; i++)
16     {
17         Console.Write(minArray[i] + " ");
18     }
19 }
20
21 static void MultipliceraMedFem(int[] minParameter)
22 {
23     for (int i = 0; i < minParameter.Length; i++)
24     {
25         minParameter[i] *= 5;
26     }
27 }

```

Resultatet av koden ovan är att första utskriften blir 1 2 3 4 5 och den andra blir 5 10 15 20 25, trots att det är parametern man jobbar med i metoden `MultipliceraMedFem` och ingen retur-sats finns.

6.5 Överlagrade metoder

En metodsignatur måste vara unik men det betyder inte att två metoder inte kan ha samma namn. Bara de inte har exakt samma parametrar. Signaturen utgörs av hela metodhuvudet, inklusive returtyp och parametrar. Det är t.ex. helt ok att i ett och samma program ha två metoder enligt nedanstående:

```

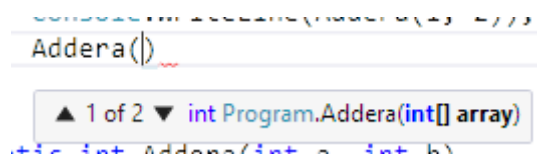
1 public static void Main()
2 {
3     int[] tal = { 1, 2 };
4     Console.WriteLine(Addera(tal));
5     Console.WriteLine(Addera(1, 2));
6 }
7
8 static int Addera(int a, int b)
9 {
10     return a + b;
11 }
12

```

```

13 static int Addera(int[] array)
14 {
15     int summa = 0;
16     for (int i = 0; i < array.Length; i++)
17     {
18         summa += array[i];
19     }
20     return summa;
21 }
    
```

Metoder med samma namn men olika signatur kallas *överlagrade metoder* och det är dessa du kan bläddra mellan (med piltangenterna) i Visual Studio när du börjar skriva en parantes efter metodnamnet i ett anrop.



Figur 6.2: Bläddra bland överlagrade metoder.




6.6 Gemensamma variabler

Om en variabel behöver vara gemensam för flera olika metoder (dvs. åtkomlig från vilken metod som helst) kan man deklarera den precis ovanför *Main*-metoden. Denna variabeldeklaration ska föregås av **static** av samma anledning som angavs tidigare. Ett minimalt exempel kan se ut såhär:

```

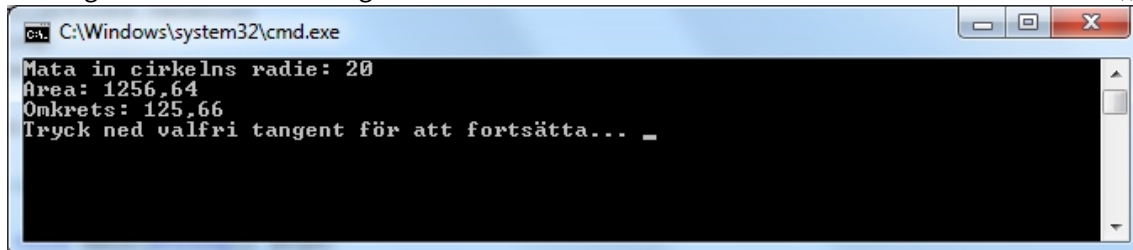
1 static int a;
2
3 public static void Main()
4 {
5     a = 12;
6     SkrivUtA();
7 }
8
9 static void SkrivUtA()
10 {
11     Console.WriteLine(a);
12 }
    
```


6.7 Att göra

-  Läs mer om metoder på csharpskolan.se/article/metoder-och-funktioner.
-  Mer information om metoder finns i boken på s. 142-161. Du kan hoppa över delen om referens- och outparametrar.
- ☒ Genomför självtestet *Metoder* på Moodle: moodle.teed.se.
-  Genomför övningarna 38-47 här nedan.

6.7.1 Övningar

38. Skriv ett program som låter användaren mata in ett heltal och sedan svarar med kvadraten på det talet. Beräkningen skall göras i en egen metod som returnerar kvadraten.
39. Redigera i ditt program från uppgift 38 så att användaren får bestämma både bas och exponent. Programmet svarar sedan med värdet av potensen (som beräknas i en egen metod).
40. Skapa ett program där du matar in radien på en cirkel. Programmet skall sedan räkna ut både area och omkrets på cirkeln. Till beräkningarna skall du använda dig av metoder, en för areabereäkningen och en för beräkningen av omkretsen. Runda av till två decimaler med `Math.Round()`.



```

C:\Windows\system32\cmd.exe
Mata in cirkelns radie: 20
Area: 1256,64
Omkrets: 125,66
Tryck ned valfri tangent för att fortsätta... _
    
```

41. Skriv ett program där du definierar en heltalsarray. Skapa sedan en egen metod som summerar alla talen i arrayen. Metoden skall returnera ett heltal. Skriv ut summan.
42. Skapa en tecken-array med de sex första bokstäverna i alfabetet som gemener. Skapa sedan en metod som tar emot en char-array och byter från gemener till versaler (studera Unicode-tabellen om du glömt av hur man gör). Metoden skall ha returtypen `void`. Skriv ut samtliga tecken i arrayen med hjälp av ännu en metod av returtypen `void`.
43. Skriv ett program där tre endimensionella heltalsarrayer av samma storlek skapas. Fyll två av dessa med slumpade heltal mellan 1 och 10.
 - a) Skriv sedan en metod som tar emot tre heltalsarrayer `a`, `b` och `c`. Metoden skall beräkna summan för respektive element i `a` och `b` för att sedan lagra detta i `c` (den tomma arrayen). Metoden skall vara generel på så vis att den fungerar för vilken arraystorlek som helst (i en dimension).

- b) Skriv sedan till metod som tar emot en heltalsarray och skriver ut samtliga av dessa tal till skärmen på en och samma rad. Metoden skall inte returnera något (**void**).
 - c) Använd dina metoder för att beräkna summa och sedan göra utskrift av samtliga tre arrayer.
- 44. Gör ett återbesök på det första praktiska provet på Moodle: moodle.teed.se. Ta en titt på del 2 av temperaturuppgiften (då man ska kunna mata in 3 temperaturer och beräkna medel). Lös denna med dina nya kunskaper om arrayer och metoder!
- 45. I rollspel brukar tärningsslag beskrivas som "1T6" vilket betyder att en tärning med 6 sidor skall kastas en gång. Ibland skall en tärning kastas flera gånger t.ex. "3T6" vilket betyder 3 kast med en vanlig tärning. Skriv ett program där du skapar en metod `KastaTarning` som har en parameter för hur många gånger tärningen skall kastas. Resultatet skall returneras. Vi förutsätter en vanlig 6-sidig tärning.
- 46. Bygg vidare på övning 45 och skapa en överlagrad metod till `KastaTarning`. Den nya varianten skall ha både antalet sidor på tärningen som skall kastas samt antalet gånger tärningen skall kastas som parametrar. Dvs. du skall kunna slå "3T6" eller "2T20" om du så vill med att anropa `KastaTarning(6, 3)` eller `KastaTarning(20, 2)`. Resultatet skall givetvis returneras från metoden.
- 47. Bygg vidare på övning 46 och skapa ytterligare en överlagrad metod till `KastaTarning`. Denna gång skall parametern vara av typen `string`. Du skall nämligen direkt kunna tolka texten "3T6", "2T4" eller "1T20" som ett tärningskast och returnera resultatet. Här krävs lite stränghantering, kolla i tidigare kapitel om du är osäker!