

C#

Visual Studio 2010

Kapitel 2
Variabler och operatorer

Innehåll

2.1 Inledning	1
2.2 Reserverade ord	1
2.3 Variabler	2
2.4 Datatyper	3
int	3
long	4
float	4
decimal.....	4
char	4
2.5 Operatorer	7
Aritmetiska operatorer	7
Tilldelningsoperatorer	7
Relationsoperatorer	8
Logiska operatorer	8
2.5 Sanningstabell	9
2.6 Typomvandlingar	9
Cast.....	9
Parse	10
Convert.....	10

2.1 Inledning

I det här kapitlet tittar vi på programmeringen mest grundläggande funktioner som ger oss möjlighet att lagra och bearbeta data.

Beräkningar utför vi med operatörer som omfattar de fyra räknesätten plus, minus, multiplikation och division. operatörer.

Exempel 1:

```
x = 3 + 4; // variabeln x tilldelas värdet 7
```

Observera att i programmering säger man inte att x är lika med 7 utan man säger att variabeln x **tilldelas** värdet 7.

Exempel 2:

```
x = x + 4; // om variablen x från början har värdet 7 som i föregående  
//exempel så tilldelas nu x värdet 11 (7 + 4).
```

Av ovanstående två exempel kan vi klart se att vi måste tolka likhetstecknet annorlunda i programmeringen jämfört med matematiken. Det vore ju orimligt att påstå att 7 skulle vara lika med 11 som i det andra exemplet. Vi kan dock lugnt påstå att variabeln x tilldelas det nya värdet 11.

2.2 Reserverade ord

Alla programmeringsspråk består av ett antal reserverade ord. Var och ett av dessa reserverade ord har en speciell betydelse som anropar olika funktioner i den skrivna koden. Att de kallas reserverade ord betyder helt enkelt att de inte får användas i några sammanhang förutom till det de är avsedda för.

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Förutom de reserverade orden finns det ytterligare ett antal så kallade kontextuella ord som kan omdefinieras vid avancerad programmering. Normalt används de som reserverade ord.

<code>add</code>	<code>alias</code>	<code>ascending</code>	<code>by</code>	<code>descending</code>
<code>equals</code>	<code>from</code>	<code>get</code>	<code>global</code>	<code>group</code>
<code>into</code>	<code>join</code>	<code>let</code>	<code>on</code>	<code>orderby</code>
<code>partial</code>	<code>remove</code>	<code>select</code>	<code>set</code>	<code>value</code>
<code>var</code>	<code>where</code>	<code>yield</code>		

2.3 Variabler

En variabel kan liknas med en adress till ett lagringsutrymme i minnet. Vi kan ge våra variabler valfria namn även om det finns vissa regler och praxis som man bör följa.

- Ett variabelnamn får inte börja med en siffra.
- Siffrorna 0 till 9 kan dock användas i en annan position i variabelnamnet.
- Använd endast anglosaxiska alfabetet a till z respektive A till Z.
- Ett variabelnamn får alltså inte innehålla svenska tecknen ÅÄÖ eller äö.
- Observera att programspråket skiljer på stora och små bokstäver.
- Börja alltid ett variabelnamn med en liten bokstav.

När man namnger variabler använder man vanligtvis vad som kallas camelCase notation. Det vill säga att man börjar ett variabelnamn med en liten bokstav och låter de följande orden i variabelnamnet börja med stor bokstav. Microsoft förordar att man använder camelCase notation i koden. Det finns dock en annan variant som kallas underscore notation som vissa programmerare föredrar. Här följer ett exempel som jämför de olika skrivsätten

```
if ( thisLooksAppealing )
{
    youLikeCamelCase = true;
}
else if ( this_looks_appealing )
{
    you_like_underscores = true;
}
```

Att namnge variabler som `var1`, `var2`, `var3` och så vidare är helt intetsägande och gör att koden blir mycket svårläst. Namnge i stället variablerna efter dess funktion. Till exempel `kontoStallning`, `insattning`, `uttagsom` är bra variabelnamn som är lätta att förstå vad de används till. Namnen kanske ser lite larviga ut få svenska eftersom man inte får använda svenska tecken så många föredrar att använda engelska variabelnamn. Oavsett vilket så är det viktigt att vara konsekvent och aldrig blanda olika stilar i koden och att ge dina variabler namn som talar om vad de är avsedda för.

2.4 Datatyper

Då olika variabeltyper kräver olika stort minnesutrymme måste man ange vilken datatyp det gäller för att rätt minnesmängd reserveras och att den ska tolkas på rätt sätt under programkörningen.

All data lagras i bitar som antingen 0:or eller 1:or där varje datatyp kräver ett visst minnesutrymme. Om vi tittar på en av de mest använda datatypen `int` som är en förkortning av engelskans Integer vilket betyder heltal på svenska så ser vi i tabellen att den kräver 32 bitars lagringsutrymme (4 Byte). I det binära talsystemet motsvarar detta talen mellan $-2,147,483,648$ to $2,147,483,647$ som i binär form skrivs i formen -2^{31} - 2^{31} -1 som visas i tabellen.

I tabellen ser du hur många bitars lagringsutrymme som krävs i minnet för att lagra de olika variabeltyperna och i sista kolumnen hur variabelerna deklarerar och tilldelas ett värde.

I tabellen finns de vanligaste datatyperna med exempel på deras egenskaper.

Datatyp	Beskrivning	Storlek(bitar)	Omfång	Exempel
<code>int</code>	Heltal	32	-2^{31} till $2^{31}-1$	<code>int count;</code> <code>count = 42;</code>
<code>long</code>	Heltal	64	-2^{63} till $2^{63}-1$	<code>long wait;</code> <code>wait = 42L</code>
<code>float</code>	Decimaltal	32	$\pm 1,5 \times 10^{45}$ till $\pm 3,4 \times 10^{38}$	<code>float z;</code> <code>z = 0.42F;</code>
<code>double</code>	Decimaltal	64	$\pm 5,0 \times 10^{-324}$ till $\pm 1,7 \times 10^{308}$	<code>double y;</code> <code>y = 0,42;</code>
<code>decimal</code>	Valuta	128	128 valuta-tecken	<code>decimal summa;</code> <code>summa = 0.2M;</code>
<code>string</code>	Texter	16 per tecken	Ingen gräns	<code>string namn;</code> <code>namn = "Lena";</code>
<code>char</code>	Tecken	16	0 till $2^{16}-1$	<code>char lan;</code> <code>lan = 's';</code>
<code>bool</code>	Boolean	8	true eller false	<code>bool slut;</code> <code>slut = true;</code>

De olika datatyperna beskrivs mer utförligt i den följande texten.

`int`

Integer(Heltal) är den mest använda numeriska datatypen. Används till exempel i alla typer av räknare där man vill att ett kodavsnitt ska upprepas ett visst antal gånger. Likaså om man vill generera ett slumpstal för att simulera ett tärningskast vill man naturligtvis helst ha svaret som ett heltal.

```
int summa;           // Variabeln summa deklarerar som en integer
summa = 35;         // Variabeln summa tilldelas värdet 35,

for (int i=0; i < 10; i++) // En räknare som ger variabeln i
                        //värde från 0 till 9
```

long

Har man behov av att använda sig mycket stora tal större än 2 miljarder som är gränsen för Integer, kan man använda datatypen `long` som också representerar heltal men med högsta gräns på 9 trillioner (en millon millioner millioner). Använd inte denna datatyp i onödan då den tar upp dubbelt så mycket minne jämfört med vanliga heltal,

```
long ettMycketLitetExtremtal; // Variabeln deklarerar och
ettMycketLitetExtremtal = 40564L; // tilldelas ett värde
```

Här är C# lite speciellt då man måste avsluta taltilldelningen med ett "L" och skulle hoppa över detta så tolkas talet 40564 som en Integer. Samma princip gäller även för datatyperna `float` och `decimal`.

float

Floating-point numbers. Denna datatyp finns ingen större anledning att använda då avrundningsfelen är större jämfört med `double`. En `double` kräver i och för sig mer minnesutrymme, men det är marginellt i detta sammanhang.

```
float dagensYttertemperatur; //Tilldelningen av värdet måste
dagensYttertemperatur = 7.5F //avslutas med ett F.
```

decimal

Eftersom datorer arbetar med ettor och nollor så sker alla beräkningar enligt det binära talsystemet så innebär det ofrånkomligt avrundningsfel i vissa fall när resultatet ska redovisas i ett decimalt format. Även om felet kanske bara är 18 öre på flera hundratusentals kronor så kommer bokföringens balans inte att stämma.

Datatypen `decimal` har ett mindre talomfång än `double` men i gengäld mycket högre precision och eliminerar risken för avrundningsfel. När det gäller ekonomiska system rekommenderas denna datatyp.

```
decimal inlaningsRanta;
inlaningsRanta = 1.756M
```

char

Denna datatyp rymmer ett tecken. Bra att använda om man vill fråga om en användare vill fortsätta med ett program eller inte och låta denne svara `J` eller `N`.

```
char svar; //Exemplet är skrivet i pseudokod.
om svar == 'J' //Ett likhetstecken betyder ju tilldelning
    starta om //medan dubbla likhetstecken kontrollerar
om svar == 'N' //om ett svar är sant. J eller N
    avsluta //Datatypen char ska omges av två enkla
//citattecken. På dataslang "enkelfnuttar"
```

Datotypen `char` kan alltså endast innehålla ett enda tecken och faktum är att om man skulle använda datotypen `string` i ovanstående exempel så skulle det fungera på samma sätt. Praxis är dock att man använder sig av typen `char` när man förväntar sig att användaren ge ett enteckensvar.

Varje tecken i datotypen `char` består av 16 bitar vilket motsvaras av ett siffervärde mellan 1 och 65355 som översätts ett tecken enligt en tabell som kallas UNICODE som täcker hela världens nationella teckenuppsättningar. Att jämföra med den gamla standarden ASCII (American Standard Code for Information Interchange) som endast rymde 256 tecken och det är på grund av denna begränsade teckenuppsättning som ligger som grund för de flesta programspår som är orsaken till att vi inte kan använda svenska tecken i vår kod.

Däremot kan vi skriva ut vilka tecken som helst eftersom utskriftsfunktionerna följer UNICODE-standarderna.

Ett sätt att skriva ut ett tecken är att ange dess decimala värde och omvandla detta till ett tecken som har motsvarande kod.

```
char mittTecken1; //Riktig kod!
mittTecken1 = ((char) 65); //vi måste tala om att talet 65
textBox.Text = (mittTecken1); //ska tolkas som UNICODE
```

Ett A skrivs ut

```
char mittTecken2; //deklaration av variabeln
mittTecken2 = ((char) 66); //tilldelning av ett värde
textBox.Text = (mittTecken2); //utskrift till en textruta
```

Ett B skrivs ut

string

Datotypen `string` används för att skriva hela textsträngar, alltså löpande text. Till skillnad från `char` så omgärdas man strängen med dubbla citattecken, på dataslang "dubbelfnuttar"

```
string minText;
minText = (" Hello world, let's Rock and Roll")
textBox.Text = (minText);
```

Syntaxen för att skriva ut en `string` är i princip densamma som för en `char` i de båda exemplen och det finns fler samband, men det kommer vi ta upp längre fram i kursen.

Om en sträng omgärdas av dubbla citattecken hur skriver man då citattecken i en sträng?

Titta på exemplet:

```
minText = ("Texten innehåller "dubbelfnuttar" plus andra ord")
```

Den här kodraden skulle innebära kompileringsfel och inte skulle gå att köra. Anledningen till detta är att delen "Texten innehåller" kommer tolkas som en textsträng

med början och slut. Det som kommer efter andra citattecknet saknar mening och kompilatorn kommer slå bakut.

Problemet går naturligtvis att lösa och lösningen är att använda ett så kallat "escapetecken". Ett tecken som talar för kompilatorn att den inte ska bry sig om att tolka det efterföljande tecknet. Escapetecknet är i form av en backslash - \ och koden kommer då att se ut på följande sätt:

```
minText = ("Texten innehåller \"dubbelfnuttar\" plus andra ord")
```

Om man då vill ha med själva tecknet backslash måste man även i detta fall använda escapetecknet. I exemplet vill man ange en sökväg till en fil i datorn.

```
minFil = C:\MinaDokument\Kompendium\Introduktion.indd FEL! FEL! FEL!
```

Kompilatorn kommer inte förstå vad som menas med escapetecknet \M. Vi måste skriva:

```
minFil = C:\\MinaDokument\\Kompendium\\Introduktion.indd RÄTT!
```

Nu kommer det fungera!

Det finns ett par escapetecken som har en speciell betydelse som man använder ofta, nämligen \n och \t som betyder radbyte (newline) och tab.

```
mittRadbyte = ("En text som jag avslutar med ett radbyte \n");
```

bool

Den här datatypen har namngivits efter George Boole 1815-1864 en matematiker och filosof som skapade den boolska algebran som är grunden i all modern datoraritmetik. Principen är egentligen väldigt enkel. En boolsk variabel är antingen sann eller falsk med värdena true eller false.

Jämför med en lampa. Är strömbrytaren till (true) så lyser lanpan och om den är av (false) så lyser inte lampan. Det är alltså bara dessa alternativ som finns.

Den här datatypen kan testa ett visst tillstånd, har någon gjort något eller inte? Men oftast används den för att kontrollera om vissa villkor är uppfyllda, till exempel att testa om ett tal ligger inom ett visst intervall eller inte. Tänk om vi vill testa om fyra är mindre än femton. Spännande!

```
int mittTal;  
mittTal = 4;  
if(mittTal < 15) // om (if) talet är mindre än (<) femton.  
textBox.Text = ("Det är sant att mittTal är mindre än femton");
```

Som du säkert märkt så använder jag olika teckenfärger i texten och jag gör det för att det är precis på detta sätt koden ser ut i Visual Studio. När jag börjar en rad med två // och använder grön text är det för att markera en kommentar. Kompilatorn hoppar över all text som börjar med dubbla snedsträck. Observera att escapetecknen som används i datatypen `string` består av bakvända snedsträck.

2.5 Operatörer

Datorer kallades från början för beräkningmaskiner som senare kallades för datamaskiner vilket förkortades till nuvarande datorer. På engelska heter det computer som kommer av ordet beräknare. Det är inte underligt att en av datorernas viktigaste funktioner är att utföra beräkningar oavsett om det handlar om rena matematiska beräkningar, krypteringar, komprimeringar, partitioneringar eller urval och sortering av databaser. Ja, i stort sätt allt handlar allt om någon form av beräkningar.

Någon form av operatörer används i princip all programkod, Utan dessa operatörer kan man inte utföra beräkningar eller jämföra olika variablers värde.

Det finns fyra huvudtyper av operatörer.

Aritmetiska operatörer

De aritmetiska operatörerna omfattar de 4 räknesätten samt operatören modulus som visar restvärdet vid heltalsdivisioner.

Ex. $16 / 11$ där 11 går en gång i 16 och resten av divisionen (modulus) blir 5.

Dividerar man däremot samma tal som flyttal blir svaret 3,2 och i detta fall saknar modulus betydelse. Alltså, modulus kan användas enbart vid heltalsdivisioner (två integer delade med varandra).

Operator	Symbol	Exempel	Resultat
addition	+	$16 + 11$	27
subtraktion	-	$16 - 11$	5
multiplikation	*	$16 * 11$	176
decimaldivision	/	$16 / 11$	1,4545...
heltalsdivision	/	$16 / 11$	1
modulus	%	$16 \% 11$	5

Tilldelningsoperatörer

Värdet på en variabel kan förändras ett flertal gånger med hjälp av tilldelningsoperatörer som visas här. Variabeln x tilldelas först värdet 42 som förändras rad för rad.

I tredje raden nerifrån blir x noll därför att $42 \text{ modulus } 6$ går jämt ut och det blir alltså inget restvärde.

Variabeln y är lika med 6 och x börjar initialt med värdet 42 och successivt ändra värde rad för rad.

Operator	Symbol	Exempel	Resultat
tilldela	=	$x = 42;$	
addera och tilldela	+=	$x += y$	$x = x + y$ (48)
sutrahera och tilldela	-=	$x -= y$	$x = x - y$ (42)
multiplitera och tilldela	*=	$x *= y$	$x = x * y$ (252)
dividera och tilldela	/=	$x /= y$	$x = x / y$ (42)
modulera och tilldela	%=	$x \% = y$	$x = x \% y$ (0)
värdeökning	x++	$x + 1$	$x = x + 1$ (1)
värdeminskning	x--	$x - 1$	$x = x - 1$ (0)

Relationsoperatorer

Denna typ av operatorer används för att jämföra två värden som antingen kan vara sanna eller falska och resultatet av detta blir något av de boolska värdena true or false. Ett enkelt likhetstecken betyder tilldelning av ett värde medan dubbla likhetstecken provar om uttrycken på vardera sidan är lika varandra.

I exemplet nedan låter vi både x och y ha värdet 20.

Namn	Operator	Exempel	Resultat
Lika med	==	<code>x == 20</code>	true
Inte lika med	!=	<code>x !=20</code>	false
Större än	>	<code>x > y</code>	false
Större eller lika med	>=	<code>x >= y</code>	true
Mindre än	<	<code>x < y</code>	false
Mindre eller lika med	<=	<code>x <= y</code>	true

Logiska operatorer

Med hjälp av logiska operatorer kan jämföra hela uttryck med varandra och man kan sammansätta ett flertal av dem till mycket komplexa villkorssatser.

I exemplet nedan låter vi nu x ha värdet 20 och y värdet 10.

Namn	Operator	Exempel	Resultat
And (och)	&&	<code>(x==20) && (y==30)</code>	false, båda uttrycken måste vara sanna
Or (eller)		<code>(x==20) (y==30)</code>	true, ett eller båda uttrycken måste vara sanna

När en villkorlig sats evalueras kontrolleras uttrycken från vänster till höger och avbryts om uttrycket blivit fastslaget som falskt. Det inte är då inte intressant att evaluera vidare och på så sätt bespara processorn onödig belastning.

För att kontrollera om ett tal (x) är större eller lika med 5 OCH om det är mindre än 24. Det vill säga ett tal mellan 5 och 23.

```
if ( x >= 5 && x < 24)
```

För att kontrollera om ett tal (x) är större än 25 ELLER att ett annat tal (y) < 10 blir detta uttryck sant om något eller båda uttrycken är sanna.

```
if( ( x > 25 || y < 10)
```

Om vi vill kontrollera om talet x ligger inom intervallen 1 till 5 eller 25 till 30.

```
if ( x > 0 && x <= 5 || ( x <= 25 && x <= 30)
```

Om vi skriver ut resultatet av ovanstående exempel får vi följande talföljd :

1 2 3 4 5 25 26 27 28 29 30

2.5 Sanningstabell

Denna tabell förutsätter att variablerna X och Y är boolska variabler som omväxlande tilldelas värdena true respektive false.

X	Y	X && Y	X Y	X ^ Y
true	true	true	true	false
true	false	false	true	true
false	true	false	true	true
false	false	false	false	false
false	false	false	false	false

Om vi tittar i de två första kolumnerna ser vi utgångsvärdena X och Y och hur resultatet utfaller om båda är sanna, ett av värdena är sant och det andra är falskt, eller vice versa eller om båda är falska.

I kolumn fem står uttrycket $X \wedge Y$ (caret eller på dataslang "tak") där uttrycken måste ha olika värden för att bli sant.

Titta på exemplet nedan som blir sant om båda villkoren är sanna. Det vill säga om variabeln `tal` har ett värde som ligger mellan 8 och 79.

```
if (tal >= 8 && tal < 80)
    x      y
```

2.6 Typomvandlingar

I vissa fall behöver man tolka en variabel som om den vore av en annan typ för att den ska passa in i vissa sammanhang. Eftersom variabler av olika typer representeras olika i minnet måste man ta till speciella metoder för att hantera detta.

Att tilldela en taltyp till en annan går dock bra om den ryms i en mindre. Det går alltså bra att tilldela ett tal av typen `double` som består av 64 bitar med ett tal av typen `integer` som består av 32 bitar då en `integer` "ryms" i en `double`. Men alltså inte tvärt om.

Exempel:

```
int i = 23;
double d = 14.75;
d = i; //ok, en integer (32 bitar ryms i en double 64 bitar värdet
//d tilldelas värdet 23 och det gamla värdet 14.75 raderas

i = d; //fel en double 64 bitar ryms inte i en integer 32 bitar
```

Cast

För att kunna tilldela ett tal av typen `double` till en `integer` måste vi göra en typomvandling (cast) och tillfälligt kapa av decimalerna i talet av typen `double` så att det ryms i en `integer`.

```
int i = 42;
double d = 3.14;
i = (int)d; //variabeln i tilldelas värdet 3 medan
//y oförändrat behåller värdet 3.14
```

Parse

I många program ber man användaren att mata in ett tal i en textruta. Eftersom de data som man matar in i denna textruta hanteras som en text. Tecknen "14" består av två tecken i en sträng en etta och en fyra och inte talet 14 som lätt man luras att tro.

För att siffrorna i en textsträng ska kunna tolkas som ett tal kan man använda metoden `parse` för att omvandla dem till talformat.

```
int a;  
double b;  
string heltal= "403";  
string decimaltal = "3.14"  
  
antaVaror = int.Parse(heltal)  
varuPris = double.Parse(decimaltal)
```

Convert

En alternativ metod till `parse` är att använda `convert`. Metoden `parse` omvandlar automatiskt text till en 32 bitars integer eller till en 64 bitars double. Med metoden `convert` kan man konvertera en textsträng med siffror till det talformat man önskar. Av denna anledning måste man alltid specificera formatet.

```
int y;  
string tal= "403";  
y = Convert.ToInt32(tal);
```