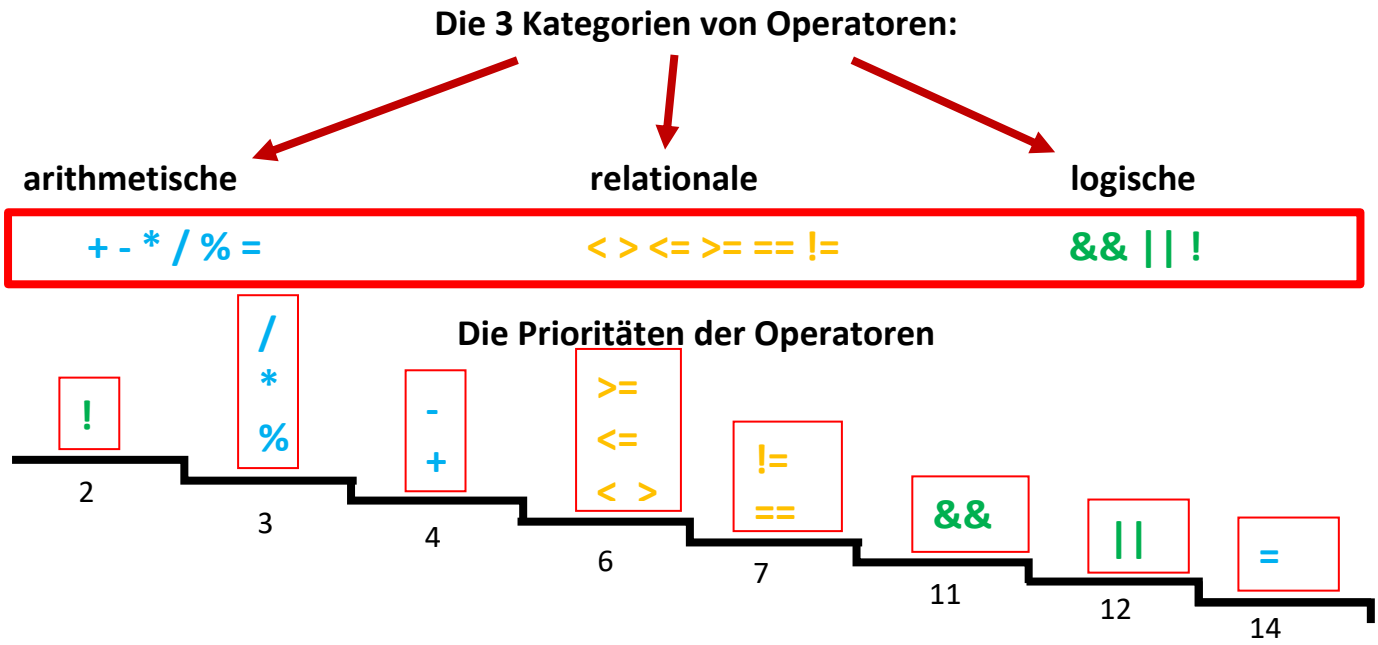
2.6.1 Operatoren in C#

Operatoren sind Funktionszeichen, mit welchen rechnerische, vergleichende oder logische Operationen programmiert werden.



Operatoren mit höherer Priorität, werden zuerst ausgewertet. Das Wort „Priorität“ könnte auch durch den Begriff „Rang“ ersetzt werden - daher 1. Rang ist höherwertiger als der 2. Rang.

Prioritäten der einzelnen Operatoren

Kategorie	Name	Symbol	Auswertreihenfolge	Priorität
Monadisch				
	Prä-Inkrement	++	Links nach Rechts	1
	Prä-Dekrement	- -	Links nach Rechts	1
	Adresse	&	Rechts nach Links	2
	Bitweises NOT	~	Rechts nach Links	2
	Typumwandlung	(typ)	Rechts nach Links	2
	Logisches NOT	!	Rechts nach Links	2
	Negation (VZ)	-	Rechts nach Links	2
	Plus-Zeichen (VZ)	+	Rechts nach Links	2
	Post-Inkrement	++	Rechts nach Links	2
	Post-Dekrement	- -	Rechts nach Links	2
Multiplikativ				
	Modulo	%	Links nach Rechts	3
	Multiplikation	*	Links nach Rechts	3
	Division	/	Links nach Rechts	3
Additiv				
	Addition	+	Links nach Rechts	4
	Subtraktion	-	Links nach Rechts	4
Bitweises Shift				
	Linksshift	<<	Links nach Rechts	5
	Rechtsshift	>>	Links nach Rechts	5
Relational				
	Kleiner als	<	Links nach Rechts	6
	Kleiner-gleich	<=	Links nach Rechts	6
	Größer als	>	Links nach Rechts	6
	Größer-gleich	>=	Links nach Rechts	6
	Gleich	==	Links nach Rechts	7
	Nicht gleich	!=	Links nach Rechts	7
Bitweise				
	AND	&	Links nach Rechts	8
	XOR	^	Links nach Rechts	9
	OR	|	Links nach Rechts	10
Logisch				
	AND	&&	Links nach Rechts	11
	OR	||	Links nach Rechts	12
Ternär				
	Bedingter Ausdruck	? :	Rechts nach Links	13
Zuweisung				
	Arithmetisch	=	Rechts nach Links	14
		+=	Rechts nach Links	14
		-=	Rechts nach Links	14
		*=	Rechts nach Links	14
		/=	Rechts nach Links	14
		%=	Rechts nach Links	14
	Shift	>>=	Rechts nach Links	14
		<<=	Rechts nach Links	14
	Bitweise	&=	Rechts nach Links	14
		|=	Rechts nach Links	14
		^=	Rechts nach Links	14

Arithmetische Operatoren

Arithmetischen Operationen benötigt man, um mit Zahlen zu rechnen.

Operator	Int	Float	Operation	Beispiel	Ergebnis
+	X	X	Vorzeichen (unäres Plus)	+7	7
-	X	X	Vorzeichen (unäres Minus)	-8	-8
+	X	X	Addition (binäres Plus)	5+6	11
-	X	X	Subtraktion (binäres Minus)	9-3	6
*	X	X	Multiplikation	8*4	32
/	X	X	Division (bei Integer-Zahlen abgeschnitten!)	1/2	0
/	X	X	Division (sobald Gleitkommazahlen dabei, als Gleitkommadivision durchgeführt)	1.0/2	0.5
%	X		Modulo (Rest bei Integer-Division, Vorzeichen des Dividenden bleibt)	-7%3	-1
=	X	X	Zuweisung (linker Seite wird der Wert der rechten zugewiesen)	a=8	8

Relationale Operatoren (Vergleichsoperatoren)

Die relationalen Operatoren führen Vergleiche durch und ergeben wie die logischen Operatoren Wahrheitswerte, d. h. 0 oder 1.

Operator	Int	Float	Operation	Beispiel	Ergebnis
<	X	X	Vergleich auf kleiner	x<y	1, wenn x kleiner y, 0 sonst
>	X	X	Vergleich auf größer	y>x	1, wenn y größer x, 0 sonst
<=	X	X	Vergleich auf kleiner oder gleich	x<=y	1, wenn x kleiner oder gleich y, 0 sonst
>=	X	X	Vergleich auf größer oder gleich	y>=x	Wie bei x<=y
==	X	X	Prüfung auf Gleichheit	x==y	1, wenn x und y gleich, 0 sonst
!=	X	X	Prüfung auf Ungleichheit	X!=43	0 wenn x gleich 43, 1 sonst

Logische Operatoren

Die letzte Klasse an Operatoren sind die logischen Operatoren. Dabei werden Integerzahlen als wahrer Wert interpretiert, wenn sie ungleich Null sind, als falscher Wert sonst. Das Ergebnis ist 1 falls wahr, 0 falls falsch. Auch hier ist eine Anwendung auf Fließkommazahlen möglich, aber nicht sinnvoll.

Operator	Int	Float	Operation	Beispiel	Ergebnis
&&	X		Logisches UND	i&&j	1, wenn i und j ungleich 0, 0 sonst
||	X		Logisches ODER	a||b	0 wenn a und b gleich 0, 1 sonst
!	X		Logisches NOT (Negation)	!q	1 wenn q gleich 0, 0 sonst

Da logische Werte in C# nichts anderes sind als Integerzahlen, kann man mit ihnen auch rechnen. Dabei kann man z. B. folgenden Trick nutzen: `a = 5 + !b * 7;`
Dieser Ausdruck weist a den Wert 5 oder 12 zu, je nachdem ob b Null ist oder nicht (falls b nicht Null, ist !b Null und !Null ist 1). Man sollte sich allerdings im Klaren darüber sein, dass solche „Tricks“ der Lesbarkeit des Programms schaden!

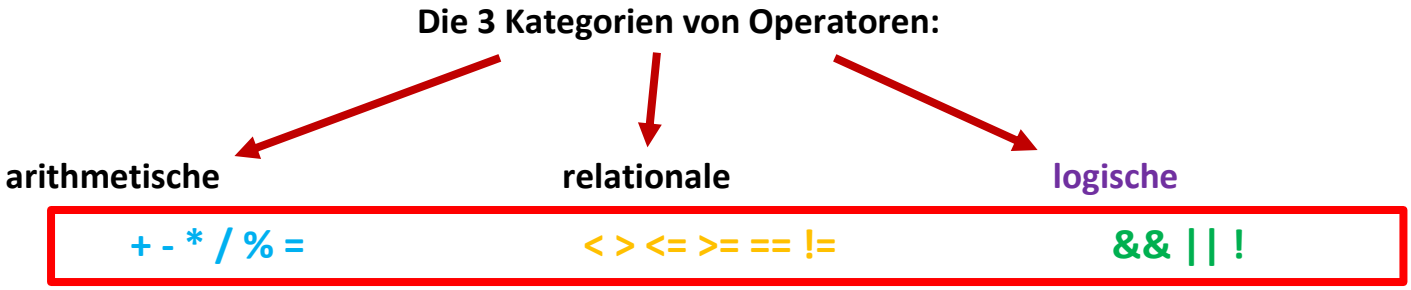
2.6.2 (Logische) Operatoren

Operatoren sind Funktionszeichen, mit welchen rechnerische, vergleichende oder logische Operationen programmiert werden. Diese Operatoren werden zum Rechnen, für Zuweisungen aber auch für Vergleiche benötigt. Insbesondere können mit den logischen Operatoren bspw. in einer if- oder while-Bedingung mehrere Überprüfungen/Vergleich miteinander verknüpft werden.

Mit Hilfe folgender Vorlage lernen Sie einige Operatoren kennen und können diese der jeweiligen Kategorie zuordnen. Öffnen sie die Vorlage:

…\Zuordnung_Operatorenkategorien.docx

…und speichern Sie sie unter dem Namen „Operatoren in C++“ für sich ab. Ordnen Sie die Operatoren per Drag and Drop den drei Operatortypen zu.



1.Bsp. arithmetische Operatoren:

```
i_zahl = 6 + 4;
```
//der + Operator sorgt dafür, dass die Summe aus 6 und 4 berechnet wird. Der = (Zuweisungs-) Operator sorgt dafür, dass der berechnete Wert 10 in der Variable `i_zahl` gespeichert wird

2. Bsp. arithmetische Operatoren

```
i_zahl = 7 % 3 + 4;
```
//der `% (modulo)` Operator sorgt dafür, dass der Rest einer Division von 7 durch 3 berechnet wird (hier: 1). Da Modulo ein Punkoperator ist, gilt „Punkt vor Strich“. Die + Operation wird also erst danach ausgeführt und die Summe 5 wird letztlich durch den `= (Zuweisungs-)` Operator in die Variable `i_zahl` geschrieben.

1.Bsp. relationale Operatoren:

```
if (i_zahl <= 5)…
```
//der `<= (kleiner gleich)` Operator sorgt dafür, dass der Wert der Variable `i_zahl` mit der 5 verglichen wird. Für alle Werte in `i_zahl` von -∞ (minus unendlich) bis einschließlich 5 wird die Bedingung `true` ergeben.

2.Bsp. relationale Operatoren:

```
if (i_zahl != 5)…
```
//der `!= (ungleich)` Operator sorgt dafür, dass der Wert der Variable `i_zahl` mit der 5 verglichen wird. Die Bedingung ergibt nur `true`, wenn in `i_zahl` nicht der Wert 5 drinsteht, also jeder andere beliebige Integerwert.

1.Bsp. logische Operatoren:

```
while (!i_zahl)…
```
//der ! Operator sorgt dafür, dass der Wahrheitswert der Variablen `i_zahl` umgedreht wird. Aus falsch wird wahr und umgekehrt. Solange also in `i_zahl` nicht genau 0 (also der einzige logische falsche Integerwert) steht, wird die Schleife nicht ausgeführt, bzw. wiederholt.

2.Bsp. logische Operatoren:

```
if (i_a==5 && i_b==3);
```
//der `&& (und)` Operator sorgt dafür, dass die if-Bedingung nur wahr ist, wenn beide Bedingunen auch wahr sind. Nur wenn in der Variable `i_a` genau der Wert 5 und in `i_b` genau der Wert 3 drinsteht, wird der Wahr-Fall des `if` ausgeführt.

2.6.3 Übungsaufgabe zu den logischen Operatoren ++ (Prä-Inkrement) und ++ (Post-Inkrement)

Welche Werte gibt das folgende Programm aus?

```
6   static void Main(string[] args)
7   {
8       int x = 5;
9       Console.WriteLine(x);          //5
10      x = x + 1;
11      Console.WriteLine(x);          //6
12      x++;
13      Console.WriteLine(x);          //7
14      ++x;
15      Console.WriteLine(x);          //8
16      Console.WriteLine(x+1);        //9
17      Console.WriteLine(x);          //8
18      Console.WriteLine(++x);        //9
19      Console.WriteLine(x);          //9
20      Console.WriteLine(x++);        //9
21      Console.WriteLine(x);          //10
```

WARUM ES SO SCHWER IST EIN PROGRAMMIERER ZU SEIN

Meine Mutter sagte:
„Bist du so lieb und gehst für mich zum Supermarkt? Kaufe dort eine Flasche Milch.
Wenn sie Eier haben, bringe sechs mit.“
Nun, ich kam mit sechs Flaschen Milch zurück.
Meine Mutter sagte:
„Warum zum Teufel bringst du mir sechs Flaschen Milch?“
Meine Antwort:
„Weil sie EIER hatten!“

2.6.4 Übungsaufgaben zu arithmetischen Operatoren

Ausdruck	Ergebnis	Anmerkung
`X = 3 / 4`	0	`int X;`
`X = 15 / 2.0`	7.5	`double X;`
`15 % 4`	3	
`3 + 5 % 4`	4	

Setzen Sie die Klammern so, dass die Auswertereihenfolge nicht verändert wird. (Setze zuerst um die Teilausdrücke mit hoher Priorität die Klammern)

```
X = ((- 4) * (++ i)) - (6 % 4)
```

Welchen Wert erhält X, wenn **`i = - 2`** ? → 2

Übungsaufgaben zu logische Operatoren

X	Y	Logischer Ausdruck	Bool'scher Wert
1	-1	`X && Y || Y >= 0`	True
0	0	`X < - 5 || !X && Y == 0`	True

Merke: Der &&-Operator hat höhere Priorität als der ||-Operator. Allerdings haben && und || eine niedrigere Priorität als die Vergleichsoperatoren (>, >=, <, <=) (vgl. Prioritätenliste).

Übungsaufgaben zu relationale Operatoren

X	Logischer Ausdruck	Bool'scher Wert
7	`X < 9 && X > - 5`	True
7	`!true && X >= 3`	False
7	`X ++ == 8 || X == 7`	False

X	Y	Logischer Ausdruck	Bool'scher Wert
6	3	`X!=Y||Y-3<=0&&X<Y+3`	True
6	-8	`X == 3 && X – 2 >= 0 || Y != 3`	True

Situationsbeschreibung Schuldenprogramm
(Übung zu Variablen/Entscheidungen/Operatoren)

Erstellen Sie ein Programm mit folgendem Funktionsumfang:

1. Einlesen des monatlichen Einkommens
2. Einlesen der monatlichen Ausgaben für bspw.
   Handy, Party, Kleidung, Schule, Auto und Wohnung

3. Berechnen der Gesamtsumme der Ausgaben
4. Vergleichen beider Summen und Ausgabe einer entsprechenden Meldung:
   a. Ob und wie viel Sie zu viel ausgegeben haben
   b. Ob und wie viel Sie am Ende des Monats übrig haben

Erweiterungen:

Sie können Ihre Ausgabeergebnisse noch verfeinern:

mehr Ausgaben als Einnahmen		**mehr Einnahmen als Ausgaben**	
über 100 € Schulden	weniger als 100 € Schulden	maximal 50 € übrig	mehr als 50 € übrig
jährliche Schulden berechnen	Ausgabe: "Noch im Rahmen"	Ausgabe der einzelnen Ausgabeposten in Prozent	mögliche Sparsumme auf ein Jahr

- Wie hoch wären Ihre Schulden nach einem Jahr?
- Wie viel Geld hätten Sie nach einem Jahr angespart?

Situationsbeschreibung Taschenrechner
(Übung zu Variablen/Entscheidungen/Operatoren)

Programmieren Sie einen Taschenrechner, der folgende Funktionen hat: Es werden zwei Zahlen eingelesen. Der Bediener soll durch Eingabe eines arithmetischen Operators **+** , **-** , **/** , ***** entscheiden können, welche Operation ausgeführt werden soll. Anschließend soll das Ergebnis und bestenfalls der komplette, gerechnete Term nochmals in der Konsole ausgegeben werden.