

## C# Elementare Sprachelemente

Name:  
Klasse:



## Inhaltsverzeichnis:

<b>1</b>	<b>Datentypen und Variablen in VC# .....</b>	<b>3</b>
<b>2</b>	<b>Konsolenausgabe und –eingabe.....</b>	<b>5</b>
<b>3</b>	<b>Casting - Typenkonvertierung .....</b>	<b>6</b>
<b>4</b>	<b>Kommentare.....</b>	<b>6</b>
<b>5</b>	<b>Ausdrücke und Operatoren (ohne Bitoperatoren) .....</b>	<b>7</b>
5.1	Arithmetische Operatoren.....	7
5.2	Zeichenkettenoperator.....	7
5.3	Zuweisungsoperatoren .....	7
5.4	Vergleichsoperatoren .....	8
5.5	Logische Operatoren .....	8
5.6	Rangfolge von Operatoren .....	8
<b>6</b>	<b>Kontrollstrukturen .....</b>	<b>9</b>
6.1	Auswahl.....	9
6.2	Wiederholungen .....	10
<b>7</b>	<b>Arrays (Felder) .....</b>	<b>11</b>

# 1 Datentypen und Variablen in VC#

**Variable** = Speicherbereich im Hauptspeicher, der mit einem Namen angesprochen wird.  
Jeder Variablen muss bei der Erzeugung (Deklaration) ein Datentyp zugewiesen werden.

**Datentyp** = bestimmt die Größe des Speicherbereiches, die Art der gespeicherten Daten (z. B. Zahl, Zeichenkette usw.) und die dafür zulässigen Operationen (z. B. + Operator für Addition bei Zahlen).

## Elementare Datentypen in VC#

bool	Wahrheitswert zur Speicherung der booleschen Werte true oder false.
char	16-Bit-Unicode-Zeichen.
byte	Ganze 8-Bit-Zahl ohne Vorzeichen, Wertebereich von 0 bis 255.
sbyte	Ganze 8-Bit-Zahl mit Vorzeichen, Wertebereich von -128 bis 127.
ushort	Ganze 16-Bit-Zahl ohne Vorzeichen, Wertebereich von 0 bis 65.535.
short	Ganze 16-Bit-Zahl mit Vorzeichen, Wertebereich von -32.768 bis 32.767.
uint	Ganze 32-Bit-Zahl ohne Vorzeichen, Wertebereich von 0 bis ca. $4,2 \cdot 10^9$ .
int	Ganze 32-Bit-Zahl mit Vorzeichen, Wertebereich von ca. $-2,1 \cdot 10^9$ bis ca. $2,1 \cdot 10^9$ .
ulong	Ganze 64-Bit-Zahl ohne Vorzeichen, Wertebereich von 0 bis ca. $18,4 \cdot 10^{18}$ .
long	Ganze 64-Bit-Zahl mit Vorzeichen, Wertebereich von ca. $-9,2 \cdot 10^{18}$ bis ca. $9,2 \cdot 10^{18}$ .
float	32-Bit-Fließkommazahl, Wertebereich ca. $-3,4 \cdot 10^{-38}$ bis ca. $3,4 \cdot 10^{+38}$ . 7 Stellen Genauigkeit.
double	64-Bit-Fließkommazahl, Wertebereich ca. $-1,7 \cdot 10^{-308}$ bis ca. $1,7 \cdot 10^{+308}$ . 15 bis 16 Stellen Genauigkeit.
decimal	128-Bit-Fließkommazahl, Wertebereich von ca. $-7,9 \cdot 10^{-28}$ bis ca. $7,9 \cdot 10^{+28}$ . 28 bis 29 Stellen Genauigkeit. Gleitkommatyp mit höherer Genauigkeit und kleinerem Wertebereich. Besonders geeignet für wissenschaftliche und finanzmathematische Berechnungen.
string	Zeichenkette, gebildet aus Unicode-Zeichen. Länge ist variabel.

## Variablendefinition und Initialisierung

Deklaration	<pre>int a;</pre> <pre>int a, b, c;</pre>
Deklaration mit gleichzeitiger Initialisierung	<pre>int zahl = 3;</pre> <pre>string wort = "Hallo";</pre> <pre>char zeichen = 'A';</pre> <pre>bool status = true;</pre> <pre>double zahl2 = 1.25;</pre> <pre>float zahl3 = 1.25f; // Dezimalzahlen muss f bekommen,</pre> <pre>                        // weil Wert (Literal) als double interpretiert wird.</pre> <p><u>Achtung:</u></p> <pre>int a, b = 3; // Wertzuweisung nur an b</pre>
Deklaration mit anschließender Initialisierung	<pre>int a;</pre> <pre>a = 3;</pre> <p><u>Achtung:</u></p> <pre>int a;</pre> <pre>a = a + 1; // Fehler, weil a vor erster Wertzuweisung verwendet wird.</pre>

## Namen von Variablen

Syntax von Namen	<ul style="list-style-type: none"> <li>• Buchstaben (a-z, A-Z), Ziffern (0-9) und "_" (Unterstrich).</li> <li>• nicht mit einer Ziffer beginnen.</li> <li>• keine Umlaute (ä, ö, ü) und weitere Sonderzeichen.</li> <li>• keine Übereinstimmung mit Schlüsselwörter von C#.</li> </ul>
Konventionen für Namen	<ul style="list-style-type: none"> <li>• sprechende Namen – verdeutlichen gespeicherten Wert.</li> <li>• klein schreiben - bei zusammengesetzten Wörtern angehängte Wörter groß beginnen lassen (z. B. anzahlAutos).</li> </ul>

**Konstante = Variable, der nur einmal ein Wert zugewiesen werden kann.**

Konstante	<pre>const float mwst = 1.19f;</pre>
-----------	--------------------------------------

## 2 Konsolenausgabe und –eingabe

<p>Ausgabe mit Console.WriteLine(), Console.Write()</p>	<p>Es werden die Methoden WriteLine() und Write() der Klasse Console verwendet. WriteLine() und Write() unterscheiden sich dahingehend, dass WriteLine() der Ausgabe automatisch einen Zeilenumbruch anhängt.</p> <p><u>Beispiel 1: einfache Textausgabe</u> Console.Write("Beispiele");                   // Beispiele</p> <p><u>Beispiel 2: Ausgabe von "</u> Console.Write("\"");                           // " (Info: \ = Escapezeichen)</p> <p><u>Beispiel 3: Ausgabe von \</u> Console.Write("\\");                         // \</p> <p><u>Beispiel 4: Ausgaben von Variablen</u> int a = 4; string s = "Hallo";</p> <p>Console.Write(s);                             // Hallo Console.Write(s + a);                        // Hallo4 (Info: + = Verkettungs-   // operator) Console.Write("\"" + s + "\"");            // "Hallo"(Info: \ = Escapezeichen,   // \" ergibt " und \\ ergibt \   // \n = Zeilenumbruch   // \t = Standardtabulator</p> <p><u>Beispiele Ausgabe mit Platzhalter / Formatangaben:</u></p> <p><b>{N, M : Format} :</b> N: Platzhalternummer beginnend mit 0 M (optionale Angabe): Breite der Ausgabe rechtsbündig, negative Zahl = linksbündig Format (optionale Angabe): diverse z. B. F zur Angabe von Nachkommastellen.</p> <p><u>Beispiel 1: (Nur) mit N (Platzhalter)</u> Console.Write("{0} mal {1}", a, s); // 4 mal Hallo</p> <p><u>Beispiel 2: mit Parameter M und Format</u> double d = 123.4567; Console.Write("{0:F2}", d);                // 123,46 (F2: 2 Dezimalstellen) Console.Write("{0,10:F2}", d);            //       123,46 (10 Stellen, rechtsb.)</p>
<p>Eingabe mit Console.ReadLine()</p>	<p>ReadLine liest ein oder mehrere Zeichen von der Tastatur ein. Das Eingabeende wird durch Drücken der Eingabetaste erkannt.</p> <p>Die Eingabe kann direkt in eine Variable vom Datentyp string gespeichert werden. Soll eine Zahl eingelesen werden, die in einer Variablen z. B. vom Typ int gespeichert werden soll muss eine Konvertierung stattfinden:</p> <p><u>Beispiele:</u> string s; int i; s = Console.ReadLine();                    // s bekommt Eingabe i = Convert.ToInt32(Console.ReadLine());   // Eingabe string muss   // konvertiert werden</p>

### 3 Casting - Typenkonvertierung

**Implizites Casting = automatische Typumwandlung:** dann möglich, wenn kein Verlust entsteht!

Umwandlung (Konvertierung)	Regel	Beispiel:
Gleiche Zahlen Datentypen	Von "klein" nach "groß"	<pre>int i = 1; double d = 2.50; float f = 2.50f; char c = 'z';  d = f; (aber nicht f = d)</pre>
Ungleiche Zahlen Datentypen	Von "ungenau" nach "genau" (Ganzzahl nach Dezimalzahl)	<pre>f = i; (aber nicht i = f);</pre>
Von Buchstabe nach int		<pre>i = c      ( z = 122 s. Ascii Tab.)</pre>

**Explizites Casting: mit dem Cast Operator:**

Ist mit Verlust verbunden, denn zu viele Nachkommastellen werden abgeschnitten!

Umwandlung (Konvertierung)	Regel	Beispiel:
Gleiche Zahlen Datentypen	Von "groß" nach "klein"	<pre>int i = 1; double d = 2.50; float f = 2.50f; char c = 'z';  f = (float) d;</pre>
Ungleiche Zahlen Datentypen	Von "genau" nach "ungenau" (Ganzzahl nach Dezimalzahl)	<pre>i = (int) f;</pre>
Von int nach Buchstabe		<pre>c = (char) i;</pre>

**Explizites Casting mit den statischen Methoden Convert.To...**

Hier gibt es weitere Möglichkeiten, die sonst nicht gehen

Umwandlung (Konvertierung)	Beispiel:
Von bool Wert auf Integer	<pre>int i = 1; double d = 2.50; bool b = true string s = "1.234";  i = Convert.ToInt32(b) // i=1</pre>
Von String auf Zahlen	<pre>d = Convert.ToDouble(s);</pre>

### 4 Kommentare

Kommentare dienen der Quellcode-Erläuterung und verbessern die Lesbarkeit und damit Wartbarkeit der Scripten. Sie werden bei der Ausführung eines Skriptes übersprungen.

```
// dies ist ein einzeiliger Kommentar

/* dies ist ein
mehrzeiliger
Kommentar */
```

## 5 Ausdrücke und Operatoren (ohne Bitoperatoren)

Ausdrücke gehören zu den kleinsten ausführbaren Einheiten eines Programms. Sie dienen dazu, Variablen einen Wert zuzuweisen, numerische Berechnungen durchzuführen oder logische Bedingungen zu formulieren.

### 5.1 Arithmetische Operatoren

Diese Operatoren führen mathematische Berechnungen durch. Sie erwarten eine Ganzzahl (Integer) oder eine Fließkommazahl als Operanden und liefern ein numerisches Ergebnis zurück.

Operator	Name	Bedeutung / Beispiel
-	negatives Vorzeichen	-a kehrt das Vorzeichen der Variablen a um.
+	Addition	a + b ergibt die Summe von a und b.
-	Subtraktion	a - b ergibt die Differenz von a und b.
*	Multiplikation	a * b ist das Produkt aus a und b.
/	Division	a / b ist der Quotient von a und b.
%	Modulo	a % b ist der Rest der ganzzahligen Division von a und b. Beispiel: 4 % 2 ergibt 0 oder 41 % 10 ergibt 1
++, --	Präinkrement, Prädecrement	++a erhöht bzw. --a verringert die Variable a um 1.
	Postinkrement, Postdecrement	a++ erhöht bzw. a-- verringert die Variable a um 1 nach der Verwendung von a

**Beachte:** Bei der Division mit ganzen Zahlen ist zu beachten, dass die Nachkommastellen abgeschnitten werden. Wenn dies nicht gewünscht ist, muss eine der beiden Zahlen eine Nachkommastelle haben.

### 5.2 Zeichenkettenoperator

Zeichenketten-Operatoren werden verwendet, um Zeichenketten miteinander zu verknüpfen. Mindestens einer der Operanden muss eine Zeichenkette sein.

Operator	Name	Bedeutung
+	Verkettung	string c; int b = 4; c = "Hallo" + b; // Hallo4

### 5.3 Zuweisungsoperatoren

Zuweisungsoperatoren werden verwendet, um einer Variablen einen Wert zu zuweisen.

Operator	Bedeutung
=	Vom Operator rechtsstehende Werte werden einer links stehenden Variablen zugewiesen.
+=	Kurzform für folgende Zuweisungen: a += b bedeutet a = a + b
-=	a -= b bedeutet a = a - b
*=	a *= b bedeutet a = a * b
/=	a /= b bedeutet a = a / b
%=	a %= b bedeutet a = a % b

## 5.4 Vergleichsoperatoren

Diese Operatoren vergleichen Ausdrücke miteinander und liefern als Ergebnis entweder wahr oder falsch. Vergleichsoperatoren werden bei der Formulierung von Bedingungen (s. Kontrollstrukturen) benötigt.

Operator	Name	Bedeutung
==	Gleichheit	a == b ergibt TRUE, wenn a und b denselben Wert enthalten
!=	Ungleichheit	a != b ergibt TRUE, wenn a und b ungleich sind.
<	Kleiner	a < b ergibt TRUE, wenn a kleiner ist als b.
>	Größer	a > b ergibt TRUE, wenn a größer ist als b.
<=	Kleiner gleich	a <= b ergibt TRUE, wenn a kleiner oder gleich ist als b.
>=	Größer gleich	a >= b ergibt TRUE, wenn a größer oder gleich ist als b.

## 5.5 Logische Operatoren

Mit logischen Operatoren lassen sich Ausdrücke logisch verknüpfen. Wie bei den Vergleichsoperatoren liefert auch die Auswertung mit logischen Operatoren ein Ergebnis vom Typ "wahr" oder "falsch".

Operator	Name	Bedeutung
!	NICHT	!a ergibt die Umkehrung des Wahrheitswertes
&&	UND	a && b ergibt nur TRUE wenn sie beide TRUE sind. Andere Kombinationen sind FALSE.
	ODER	a    b ergibt nur FALSE wenn sie beide FALSE sind. Andere Kombinationen sind TRUE
^	ENTWEDER ODER	a ^ b ergibt nur TRUE wenn einer der beiden TRUE ist. Sind beide TRUE oder beide FALSE ist der Ausdruck FALSE.

## 5.6 Rangfolge von Operatoren

Zusammengesetzte Ausdrücke werden nach bestimmten Regeln abgearbeitet bzw. errechnet. Operatoren mit höherem Rang werden vor direkt benachbarten Operatoren mit niedrigerem Rang ausgeführt. Sind zwei benachbarte Operatoren vom gleichen Rang, so werden sie von links nach rechts abgearbeitet.

Die Rangfolge der Operatoren kann grundsätzlich durch runde Klammern verändert werden. Dabei wird der in runde Klammern eingeschlossene Teil zuerst bearbeitet. Treten zwei oder mehrere runde Klammernpaare hintereinander auf, so werden sie von links nach rechts abgearbeitet. Treten die Klammernpaare verschachtelt auf, so werden sie von innen nach außen abgearbeitet.

Rangfolge	Operator	Erläuterung
1 (hoch)	()	Klammern zur Gruppierung
2	! ++ --, (int) (float) (string) (array) (object)	Logisches Nicht, Prä- und Postin- /-dekrement, Datentypumwandlungen
3	* / %	Arithmetische Operatoren
4	+ - .	Arithmetische Operatoren, Zeichenkettenoperator
5	< <= > >=	Vergleichsoperatoren
6	== === !=	Vergleichsoperatoren
7	&&	Logischer Operator
8		Logischer Operator
9	= *= +* -= /= .=	Zuweisungsoperatoren



## 6 Kontrollstrukturen

Kontrollstrukturen bestimmen den Programmablauf. Man unterscheidet zwischen: Folge (Sequenz), Verzweigung (Selektion), Wiederholung (Iteration, Schleifen) und Unterprogramme (Funktionen, Methoden).

Bei Verzweigungen und Wiederholungen steuern **Bedingungen** (logischer Ausdruck) den konkreten Ablauf. Eine Bedingung liefert als Ergebnis entweder wahr oder falsch zurück.

### 6.1 Auswahl

Bezeichnung	Allgemeine Syntax	Beispiel
<b>Bedingte Verarbeitung (if)</b>	<pre>if (Bedingung) {     Anweisung(en); }</pre>	<pre>if (kartennr == "" ) {     Console.Write("Fehler"); }</pre>
<b>Einfache Alternative (if-else)</b>	<pre>if (Bedingung) {     Anweisung(en)1; }  else {     Anweisung(en)2; }</pre>	<pre>if (note &lt;= 4) {     Console.Write("bestanden"); }  else {     Console.Write("wiederholen"); }</pre>
<b>Mehrfache Alternative (else-if)</b>	<pre>if (Bedingung 1) {     Anweisung(en); } else if (Bedingung 2) {     Anweisung(en); } // weitere else-if möglich else //kann auch fehlen {     Anweisung(en); }</pre>	<pre>if (wert == 0) {     Console.Write("Null"); } else if (wert &lt; 0) {     Console.Write ("negativ"); } else {     Console.Write ("positiv"); }</pre>
<b>Mehrfache Alternative (switch-case)</b>  switch-case wird meist verwendet, wenn der Inhalt einer Variablen überprüft werden soll.  Sie ist übersichtlicher als eine längere Folge von else-if Verzweigungen.	<pre>switch (Variable) {     case Wert1: Anweisung(en); break;     case Wert2: Anweisung(en); break;     // weitere case möglich     default: Anweisung(en); }</pre>	<pre>char zeichen = 'B';  switch (zeichen) {     case 'L': Console.Write ("Löschen"); break;     case 'A': Console.Write("Anlegen"); break;     case 'B': Console.Write("Ändern"); break;     default: Console.WriteLine("Fehler"); break; }</pre>


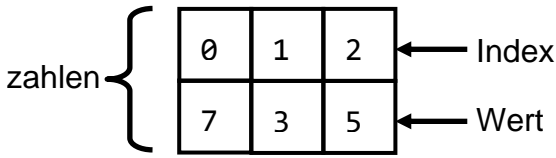
## 6.2 Wiederholungen

Bezeichnung	Allgemeine Syntax	Beispiel
<b>Kopfgesteuerte Schleife (while)</b>	<pre>while (Bedingung) {     Anweisung(en); }</pre>	<pre>int wert = -3; while (wert &lt; 0) {     Console.Write(wert);     ++wert; }  // Ausgabe: -3 -2 -1</pre>
<b>Fußgesteuerte Schleife (Do-while)</b>	<pre>do {     Anweisung(en); } while (Bedingung);</pre>	<pre>int wert = 3; do {     Console.Write(wert);     --wert; } while(wert&gt;0);  // Ausgabe: 3 2 1</pre>
<b>Zählschleife (for)</b>	<pre>for (Initialisierung;     Bedingung;     Änderung) {     Anweisung(en); }</pre>	<pre>for (int x=1; x&lt;=3; x++) {     Console.Write(x); }  // Ausgabe: 1 2 3</pre>

## 7 Arrays (Felder)

Bei den bisher verwendeten (einfachen) Variablen handelte es sich um eine mit einem Namen versehene Speicherstelle, in der genau **ein (1!)** Wert gespeichert werden kann.

Ein Array ist ebenfalls eine mit einem Namen versehene Speicherstelle, in der sich allerdings eine nahezu beliebig große Anzahl von Variablen gleichen Namens und gleichen Datentyps befinden. Unterschieden werden die einzelnen Elemente (Variablen) nur anhand einer Indizierung.

Variable	Speicherstelle im Hauptspeicher
<u>einfache Variable:</u> <code>int zahl = 4;</code>	
<u>Array Variable:</u> <code>int[] zahlen = new int[3] {7,3,5};</code>	 <p>Array Variable: zahlen            Einzelvariable des Arrays: <code>zahlen[0]</code> // Wert = 7  <code>zahlen[1]</code> // Wert = 3  <code>zahlen[2]</code> // Wert = 5</p>

**Durch die Verwendung von Arrays ergeben sich einige Vorteile:**

- Anstatt eine Vielzahl von gleichartigen Variablen zu definieren, benötigt man nun nur noch ein (1!) Array.
- Sollen für alle einzelnen Werte eines Arrays dieselben Anweisungen ausgeführt werden, kann dies mithilfe einer Schleife komfortabel durchgeführt werden, weil anstelle des Index eine Variable stehen kann.  
Beispiel: `for (int i = 0; i < 3; i++) { Console.WriteLine(zahlen[i]); }`
- Ein weiterer Vorteil ist, dass zahlreiche Methoden für die Verarbeitung von Arrays existieren. Z. Bsp. kann ein Array einer Sortiermethode übergeben werden, um die einzelnen Werte in dem Array zu sortieren.
- Es kann jederzeit auch mit der Einzelvariable wie mit einer "einfachen" Variablen gearbeitet werden z. Bsp. `zahlen[1] = zahlen[0] + 3;`

### Arraydeklaration und Initialisierung

Deklaration (1)	<code>int[] a; a = new int[5];</code> //Dimension/Größe des Arrays = 5.
Deklaration (2)	<code>int[] a = new int[5];</code>  <u>Achtung:</u> Elemente sind mit 0 vorinitialisiert!  Arrays werden in VC# als Objekte angesehen. Das Schlüsselwort <code>new</code> kennzeichnet die Erzeugung eines Objektes. Die Anzahl der Array-Elemente, die erzeugt werden sollen, wird in eckigen Klammern angegeben. Der Index beginnt mit 0. Das Array <code>a</code> enthält also die 5 Elemente (Variablen) <code>a[0]</code> bis <code>a[4]</code> .
Deklaration mit Initialisierung	
(1)	<code>int[] a = new int[3] {34,5,67};</code> //Angabe mit Größe und Werten
(2)	<code>int[] a = new int[] {34,5,67};</code> //Angabe mit Werten (legt Größe fest)
(3)	<code>int[] a = {34,5,67};</code> //Angabe mit Werten (new kann fehlen)

### Zugriff auf Array Elemente

Beispiel (1)	int[] a = new int[5] {2,5,77,3,3};  a[1] = 4; // Wert an Position 1 wird überschrieben.
Beispiel (2)	Console.WriteLine(a[3]); // Wert an Position 3 wird ausgegeben.  <u>Achtung:</u> das Array selber, also a kann nicht so ausgegeben werden. Siehe unten Schleifen mit Arrays).
Beispiel (3)	int i = 4; a[i] = 44 // Wert an Position 4 wird überschrieben. a[i-2] = 55; // Wert an Position 2 wird überschrieben.  <u>Achtung:</u> Anstelle des Index kann eine Variable oder ein Ausdruck stehen, der eine Ganzzahl zurückliefert. Wichtig: Array Grenzen dürfen nicht überschritten werden (bei Array a liegt die Grenze, d. h. der größte Index bei 4).

### Arrays und Schleifen

Beispiel (1)	int[] a = new int[5] {2,5,77,3,3};  int i ; for (i = 0; i < 5; i++) { Console.Write(a[i]); //Index als Variable }
Beispiel (2)	foreach (int x in a) { Console.Write(x); //Ausgabe des Wertes <del>x = x + 2;</del> //FEHLER: x darf nicht verändert werden. }
	<u>Hinweis:</u> Auch while oder do while Schleife mit Arrays möglich.

### Methoden und Eigenschaften für Arrays (Beispiele mit Length, und Array.Sort())

Beispiel (1)	int[] a = new int[5] {2,5,77,3,3};  Console.WriteLine(a.Length); // Größe des Arrays wird ausgegeben.  <u>Hinweis:</u> Length ist eine Eigenschaft des Arrays a.
Beispiel (2)	Array.Sort(a); //Array Elemente werden aufsteigend im Array sortiert.  <u>Hinweis:</u> Array.Sort ist eine statische Methode der Klasse Array.