Grundlegende Datentypen

(Eine Zusammenfassung aus ihrem Herdt-Buch C# - Grundlagen der Programmierung mit Visual Studio 2019, Kap. 6.6 bis 6.9)

Datentypen

Damit ein Computerprogramm eine Eingabe speichern kann, muss eine Variable erstellt werden. Einer Variable wird vom Betriebssystem ein Platz im Arbeitsspeicher zugewiesen. Je nach Art der zu speichen-den Information benötigen wir mehr oder weniger Platz. Auch können die Struktur und der Aufbau der 0 und 1 Informationen ganz unterschiedlichen Typs sein (z.B. UniCode vs. Integer oder FloatingPoint Format). Dies alles regelt der Datentyp. Eine Variable basiert bei uns immer auf einem Datentyp.

```
int groesse = 12;
```

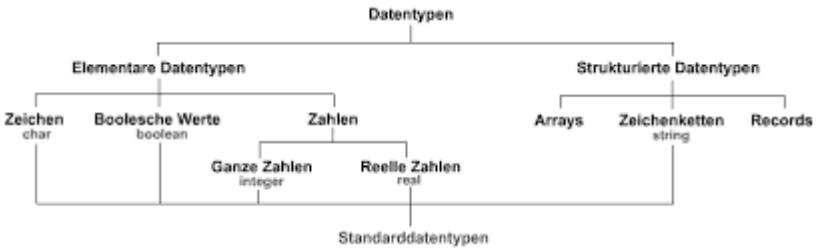
Die obenstehende Anweisung definiert eine Variable des Typs Integer mit dem Namen groesse. Variablen des Typs Integer können nur ganze Zahlen im Bereich von -2’147’483’648 bis +2'147'483’647 speichern.

Man unterscheidet folgende Hauptkategorien von Datentypen:

- Zahlen (numerische Daten, z. B. 15 oder 120.98 oder -36.50),
- Zeichen (alphanumerische Daten, z. B. «Meier)
- Boolesche (logische Daten, z. B. true oder false)
- Datumsangaben (z. B. 03.12.2025).

Numerische Datentypen

Die numerischen Datentypen werden dann benötigt, wenn im Programm mit Zahlenwerten gearbeitet werden soll. Es werden Integer-Datentypen (ganze Zahlen) und Gleitkomma-Datentypen (gebrochene Zahlen) unterschieden.



Datentypen, um ganze Zahlen zu speichern:

Datentyp	Wertebereich	Speichergrösse
sbyte	-128 bis +127	1 Byte
short (int16)	-32768 bis +32767	2 Byte
int (int32)	-2'147'483'648 bis +2'147'483'647	4 Byte
long (int64)	-9'223'372'036'854'775'808 bis +9'223'372'036'854'775'807	8 Byte
byte	0 bis +255	1 Byte
ushort (uint16)	0 bis +65'535	2 Byte
uint (uint32)	0 bis +4'294'967’295	4 Byte
ulong (uint64)	0 bis +18'446'744'073'709'551'615	8 Byte

Datentypen, um gebrochene Zahlen zu speichern:

Datentyp	Wertebereich	Genauigkeit	Speichergrösse
float (Single)	~ +/- 1.4 x 10-45 bis 3.4 x 10^{38}	7-8 Stellen	4 Byte
double (Double)	~ +/- 4.9 x 10-324 bis 1.8 x 10^{308}	15-16 Stellen	8 Byte

Der Datentyp decimal (Decimal) besitzt eine hohe Genauigkeit (28 Nachkommastellen) und ist vor allem für Finanz- und Währungsberechnungen geeignet, da Rundungsfehler seltener auftreten als bei double oder float. Dennoch sind Rundungsprobleme nicht ausgeschlossen. Man kann diesen Datentyp als ganze Zahl oder als Zahl mit festen Nachkommastellen einsetzen.

Datentyp	Wertebereich	Genauigkeit	Speichergrösse
decimal (Decimal)	0 +/- 79'228'162'514'264'337'593'543'950’335	28 Stellen	16 Byte
	+/- 1.0 x 10-28 bis 7.9 x 1028		

Hinweis: Ab dem .NET Framework 4.0 wurden zum Rechnen mit sehr grossen sowie mit komplexen Zahlen die Typen BigInteger und Complex bereitgestellt. Allerdings handelt es sich hierbei nicht mehr um primitive (grundlegende) Datentypen.

Zeichen Datentypen

Zeichen-Datentypen in C# können beliebige Zeichen oder Zeichenketten des sogenannten Unicode-Zeichensatzes enthalten.

Datentyp	Wertebereich	Speichergrösse
char (Char)	Ein Unicode-Zeichen	2 Byte
string (String)	Mehrere Unicode-Zeichen	Max. 4 GB

Hinweis: Achtung!! Unterschied zwischen 'a' und "a" erkennen!

- *\t = Tabulator*
- *\n = Zeilenumbruch*
- *\u = Spezialzeichen (z.B. \u0045 = E)*

Reguläres String-Literal	*Verbatim String-Literal*
"c:\\Daten\\M319"	*@"c:\\Daten\\M319"*
"Klicke auf \"GO\" "	*"Klicke auf ""GO"" "*

Boolescher (logischer) Datentyp

Zur Repräsentation von Wahrheitswerten (wahr bzw. falsch) dient der Datentyp bool. Ein Wahrheitswert kann nur die Literale true oder false als Wert annehmen.

Datentyp	Wertebereich	Speichergrösse
bool (Boolean)	true, false	1 Byte

Datumsangaben

Datumsangaben lassen sich mit dem Datentyp DateTime speichern. Hier handelt es sich allerdings nicht mehr um einen primitiven (grundlegenden) Datentypen. Das erkennt man auch an der Grossschreibung. Intern wird das Datum als ganze Zahl gespeichert.

Datentyp	Wertebereich	Speichergrösse
DateTime	1. Jan. 0001 bis 31.Dez. 9999, 0:00:00 bis 23:59:59	8 Byte

Hinweis: Datumsangaben können Sie nicht direkt durch eine Wertzuweisung eingeben:

- *DateTime datum1 = Convert.ToDateTime("01.01.2020");*
- *DateTime datum2 = DateTime.Parse("01.01.2020");*

Benutzereingaben (string) in andere Formate konvertieren

Die Funktion ReadLine() der Klasse Console liefert als Rückgabewert immer einen String zurück.

Da man mit Zeichenketten (String) nicht rechnen kann, muss man diese Informationen in einen passenden Aufbau umformatieren. Exemplarisch wählen wir hier den Datentyp double.

Im untenstehenden Beispiel gibt der Benutzer 20 ein. Zuerst wird der ganze Aufruf von ReadLine() ersetzt durch die Zeichenkette «20». Das heisst Codierung des Symbols «2» (= 50) plus Codierung des Symbols «0» (= 48) plus Codierung des unsichtbaren String-Schlusszeichens.

Danach wird diese Information (also der String «20») an die ToInt32()-Funktion der Klasse Convert als Parameter übergeben. Diese Funktion wandelt die Codierung in das binäre Zahlensystem um («20» > 2010 > 00000000 00000000 00000000 000101002)

```
uint age = Convert.ToUInt32(Console.ReadLine());
uint age = Convert.ToUInt32(      "20"       );
uint age =                  20                ;
```

Variablen erstellen, implizites und explizites Casting

Die Deklaration von Variablen wird mit dem Namen des Datentyps eingeleitet. Anschliessend folgt der Variablenname (identifier = Bezeichner). Mehrere Variablen desselben Typs können Sie in einer Anweisung definieren. Die Variablennamen werden dabei durch Kommata getrennt. Lokale Variablen beginnen üblicherweise mit einem Kleinbuchstaben.

```
type1 identifier1 [, identifier2 …];
```

Beispiele für Variablendeklarationen:

```
int zahl;
double count, weight, price;
char sign;
```

```
int &number;
//geht nicht, weil der Bezeichner nicht mit _ oder a..Z beginnt

double Price per Article;
//geht nicht, weil der Bezeichner keine Leerzeichen enthalten darf
```

Implizite und explizite Typkonversion

Der Compiler nimmt nicht in jedem Fall eine automatische Konvertierung von einem zum anderen Daten-typ vor. Dies geschieht nur, wenn Sie z.B. den Inhalt einer int-variablen einer float-Variablen zuweisen. Man spricht dabei von einer impliziten Typkonversion. Implizite Typkonversion geschieht nur, wenn ein Umwandeln in einen kompatiblen Typen betreffend Wertebereich und Genauigkeit und mit genügend Speichergrösse erfolgt.

Implizite Typkonversion	
von Typ	**zu Typ**
byte	short, char, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	float, double
float	double

Hinweis: Bei zuweisungskompatiblen Datentypen führt C# automatisch eine implizite Typkonversion durch. So können Sie beispielsweise eine Ganzzahl (Typ int) problemlos einer Variablen vom Typ double zuweisen. Sind die Datentypen nicht zuweisungskompatibel, so können Sie eine explizite Typkonversion durchführen. Die explizite Typkonversion wird auch als Casten (engl. casting) bezeichnet.

Möglichkeiten zur expliziten Typkonversion

Die Klasse Casting

```
int zahl1 = 5000;
short zahl2 = (short)zahl1;
```

Die Klasse Convert

```
string eingabe = "6024";
int zahl = Convert.ToInt32(s);
```

Die Methoden ToString() und Parse()

```
int zahl1 = 5000;
string s = zahl1.ToString();
//entspricht … = Convert.ToString(zahl1);

string eingabe = "6024";
zahl1 = Int32.Parse(eingabe);
```

Beispiele für Typumwandlungen (Typkonversionen)

```
    int number = 4567;
    double size = 123.12;
    char ch = 'K';
    string txt;
①   number = (int)size;
②   size = 149;
③   txt = ch.ToString();
④   txt = number.ToString();
⑤   number = Convert.ToInt32("4567");
⑥   size = Double.Parse("4567,89");
⑦   size = Convert.ToDouble("4567,89");
⑧   ch = Convert.ToChar("k");
```

① Explizite Umwandlung einer Dezimalzahl in eine Ganzzahl: `number` erhält den Wert `123`
② Implizite Umwandlung einer Ganzzahl in eine Dezimalzahl: `size` erhält den Wert `149.0`
③ Explizite Umwandlung eines Zeichens in eine Zeichenkette (String): `txt` erhält den Text `"K"`
④ Explizite Umwandlung einer Ganzzahl in eine Zeichenkette: `txt` erhält den Text `"123"`
⑤ Explizite Umwandlung einer Zeichenkette in eine Ganzzahl: `number` erhält den Wert `4567`
⑥ Explizite Umwandlung einer Zeichenkette in eine Dezimalzahl: `size` erhält den Wert `4567.89`
⑦ Weitere Möglichkeit zur expliziten Umwandlung einer Zeichenkette in eine Dezimalzahl: `size` erhält den Wert `4567.89`
⑧ Explizite Umwandlung eines Strings mit **einem Zeichen** in ein Zeichen (`char`): `ch` erhält den Wert `'k'`