1 Programmieren mit C# - Erste Beispiele

1.1 Start

Am Beginn eines minimalen Computerprogramms steht ein Textfile mit dem Quellcode des Programms. Entsprechend der Konvention zur Dateinamenserweiterung (*file extension*) für den Quelltext (*source code*) eines C# Programms endet der Dateiname eines C# Quelltextes mit .cs (für C Sharp). Quelltext in C endet mit .c, Quelltext in C++ endet mit .cpp. Der Quelltext eines C# Programms wird vom Compiler in ein ausführbares Programm übersetzt, das ist üblicherweise eine Datei mit der Endung .exe. (Die Familie der Microsoft Windows Betriebssysteme unterscheidet bei Dateinamen nicht zwischen Groß- und Kleinschreibung). Für das Betriebssystem ist ein ausführbares Programm eine Datei die als Programm geladen und ausgeführt werden kann. Der Benutzer startet das Programm z.B. durch einen Doppelklick mit der Maus auf die Datei im Explorer oder startet das Programm aus der Kommandozeile mit den Namen der Datei als Kommando. Diese drei Schritte nennt man in der bevorzugten Sprache der Softwareentwickler (Englisch) Edit - Build – Run.

Seit *Kernighan* und *Ritchie*, die beiden Entwickler der Sprache C, ihr Buch *The C Programming Language* mit dem "*Hello, world*" Programm begonnen haben ist das das übliche Beispiel für das erste Programm eines Programmierkurses. Das Programm gibt einen Text am Bildschirm aus, eben den Text "*Hello, world!*".

Das ist der Quelltext eines C# Programms, das diese Aufgabe erfüllt:

```
// ConsoleOutput.cs

class ConsoleOutput
{
   static void Main()
   {
     System.Console.WriteLine();
     System.Console.WriteLine("Hello world!");
   }
}
```

Um Programme in C# zu übersetzen benötigt man einen C# Compiler, für die Ausführung des Programms muss am Rechner eine .NET Laufzeitumgebung installiert sein. Beides ist Teil des .NET Software Developer Kit (SDK) oder wird installiert, wenn sie eine Entwicklungsumgebung wie Visual Studio .NET installieren. Während die .NET SDK gratis ist (http://msdn.microsoft.com) ist Visual Studio eine Software, die man kaufen muss. Die mit der Version 2005 eingeführte Express-Edition ist allerdings auch praktisch gratis. Steht der C# Compiler zur Verfügung und ist er über den eingestellten Suchpfad erreichbar, so übersetzt folgende Kommandozeile den Quelltext in ein ausführbares Programm:

```
…>csc ConsoleOutput.cs
```

Eine Auflistung der Dateien zeigt, dass das ausführe Programm erzeugt wurde:

```
…>dir
26.09.2004  09:47               128 ConsoleOutput.cs
```

```
26.09.2004  09:51             3.072 ConsoleOutput.exe
```

Ausführung des Programms:

```
…>ConsoleOutput

Hello world!

…>
```

Der Quelltext zeigt uns einige grundlegende C# Merkmale:

Der Programmcode ist immer Teil einer Klasse (*class*). Dies ist ein Begriff aus der objektorientierten Programmierung (OOP). C# ist eine objektorientierte Programmiersprache. Die OOP zerlegt eine Programmieraufgabe in eine Menge von Objekten. Der Objektbegriff ist uns im Prinzip vertraut. Zum Beispiel ist ein Haus, eine Zeichnung, ein Dreieck oder ein Punkt ein Objekt. Der objektorientierte Ansatz ist aber auch im Umgang mit Computerprogrammen ständig sichtbar. Ein Dialog in einer Windows Applikation ist ein Objekt, ein Eingabefeld (*edit-box* oder *text-box*) ist ein Objekt und eine Befehlsschaltfläche (*button*) ist ein Objekt. Ein Objekt kann durch bestimmte Daten charakterisiert werden. Ein Haus hat z.B. einen Besitzer, eine Adresse, eine Wohnfläche, eine Gartenfläche, einen Preis usw. Ein Dreieck hat drei Seitenlängen, drei Winkel und eine Fläche und ein Punkt hat in der Ebene mindestens zwei Koordinaten. In unserem Leider verwendet man in der Literatur der OOP eine Fülle von Begriffen, die dasselbe meinen oder unter leicht unterschiedlichen Gesichtspunkten dasselbe meinen. So nennt man die Daten eines Objektes auch Attribute, Attributwerte; Datenfelder oder einfach Felder (*fields*). In C# erfolgt das Abfragen und Setzen von Datenwerten über die Eigenschaften (*properties*) des Objektes. Ein Objekt hat immer einen bestimmten Zustand (*state*). Dieser Zustand wird durch die Attributwerte beschrieben. Ein Objekt stellt in der Regel aber auch Funktionen (*functions*) oder wie man in der OOP sagt, Methoden (*methods*) zur Verfügung. Eine Methode führt meist eine Tätigkeit aus, das kann eine Interaktion mit der Umgebung sein, das kann die Änderung des Objektzustandes sein. Außerdem kann ein Objekt Nachrichten versenden, um die interessierte Umgebung über bestimmte Ereignisse zu informieren. Objekte müssen erzeugt werden und sie müssen eine eindeutige Identität haben. Ein existierendes Objekt belegt im Speicher Speicherplatz. Nicht mehr benötigte Objekte müssen wieder zerstört werden. In C# sorgt der so genannte *garbage collector* automatisch für die Entfernung nicht mehr benötigter Objekte aus dem Speicher.

Eine Klasse (class) ist eine Schablone, der Prototyp für ein Objekt. Sie beschreibt alle Datenfelder, Eigenschaften, Methoden und Ereignisse eines Objekts. Eine Klasse ist der Bauplan, die Typdefinition nach dem ein Objekt erzeugt wird. Man nennt ein Objekt auch eine Instanz einer Klasse.

Unser erstes Programm enthält eine Klasse mit dem Namen ConsoleOutput.

```
class ConsoleOutput
{

}
```

Die Klasse definiert eine Methode mit dem Namen `Main`. Im Allgemeinen kann man beim Aufruf einer Methode der Methode Daten mitgeben. Diese Daten werden in einer Klammer nach dem Namen der Methode angegeben. Die hier definierte Methode Main nützt diese Möglichkeit nicht, deshalb ist die Klammer leer. Eine Methode kann einen Wert als Resultat liefern. Auch diese Möglichkeit wird hier nicht genützt, was durch das Schlüsselwort void zum Ausdruck kommt.

```
static void Main()
```

Bleibt noch die Erklärung des Schlüsselwortes static. Normalerweise stehen die Methoden einer Klasse erst zur Verfügung, wenn auch ein Objekt erzeugt wurde. Statische Methoden stehen auch ohne Objektinstanz zur Verfügung.

Mit C# programmiert man in einem Rahmen, der für viele Standardaufgaben fertige Lösungen zur Verfügung stellt, das ist das .NET Framework. Es enthält hunderte von Klassen für sehr viele der immer wieder benötigten Aufgaben, die man in einem Programm zu lösen hat. Viele dieser Klassen oder entsprechende Objekte verwendet man für die Programmierung der Benutzerschnittstelle. Ein Beispiel für eine solche Klasse ist die Klasse `Console`. Zur leichteren und eindeutigen Identifizierung sind die Klassen des Frameworks innerhalb verschiedener Namensräume (*namespace*) definiert. Die Klasse `Console` gehört zum Namensraum `System`.

Mit

```
System.Console.WriteLine("Hello world!");
```

wird die Methode `WriteLine` der Klasse `Console` aus dem Namensraum `System` aufgerufen. Diese Methode schreibt die Zeichenkette "Hello world!" auf den Bildschirm.

1.2 Visual Studio Entwicklungsumgebung - Zweites Programmbeispiel

1.2.1 Programm erstellen

Das Gerüst des zweiten Programmbeispiels lassen wir uns von der Entwicklungsumgebung (Visual Studio .NET) erzeugen. Wir wählen File → New → Project [Ctrl+Shift+N] und wählen dann in der Liste der Visual C# Projekte die Vorlage (*template*) *Console Application*. Als Projektname wählen wir Hello2

Das erzeugte Programmgerüst enthält gegenüber unserem ersten Programm einige zusätzliche Zeilen.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Hello2
{
   class Program
   {
     static void Main(string[] args)
     {
     }
   }
}
```

Die Zeile

```
using System;
```

erspart uns eine Menge Schreibarbeit bei der Verwendung von Methoden aus dem Namensraum System. Wir können das einleitende System. immer weglassen, wenn sich dadurch keine Namenskonflikte ergeben. Die weiteren using Zeilen sind eine Vorbereitung auf weitere oft verwendete Komponenten des .NET Frameworks.

```
namespace Hello2;
```

umgibt die Klasse `Program` mit einem Namensraum `Hello2`. Der Klassenname `Program` wird von der Entwicklungsumgebung vergeben.

`string[] args` würde in einer nummerierten Liste von Zeichenketten jene Argumente enthalten, die man dem Programm beim Aufruf mitgegeben hat.

Mit jeweils zwei Schrägstrichen (*slash*) beginnt eine Kommentarzeile. Eine andere Möglichkeit für Kommentare ist die Verwendung von `/*` als Beginn eines Kommentars und `*/` für das Ende des Kommentars.

Das fertige zweite Programmbeispiel:

```
using System;

namespace Hello2
{
   class Program
   {
     static void Main(string[] args)
     {
       string userName;       // to store the users name
       string message;        // output message

       Console.WriteLine("Hello, what's your name?");
       userName = Console.ReadLine();
       message = "Hello " + userName +
         ", nice to meet you! Have fun with C#!";
       Console.WriteLine(message);

       Console.WriteLine
         ("Press <Enter> to terminate this program!");
       Console.ReadLine();
     }
   }
}
```

Das Programm beginnt mit der Ausgabe des Textes "Hello, what's your name ?" und verwendet dann die Methode `ReadLine` des Konsolenobjekts um die Eingabe des Benutzers zu lesen und sich die Eingabe als `userName` zu merken. In einer höheren Programmiersprache kann man mit Daten sehr einfach umgehen. Für alle Daten, die man benötigt, vergibt man Namen und legt den Datentyp der Daten fest. Der passende Datentyp für eine Zeichenkette (also einfach einen Text) hat in C# den Namen `string`. Im Programm werden also zwei Datenfelder (man spricht auch von Variablen) `userName` und `message` vom Typ Zeichenkette vereinbart. Die Vereinbarung muss erfolgen, bevor man die Daten im Programm verwendet.

Betrachten wir noch die beiden Zeilen

```
Console.WriteLine("Hello, what's your name?");
Console.WriteLine(message);
```

Der Editor zeigt uns als Tooltipp an, welche Varianten der Console-Methode `WriteLine` zur Verfügung stehen. Wir verwenden die Variante:

```
void Console.WriteLine(string value)
```

Wie ist das zu lesen? Die Methode benötigt ein Argument `value` vom Typ `string`, das ist die Zeichenkette, die ausgegeben wird. Das `void` am Beginn bedeutet, dass die Methode keinen Rückgabewert hat. Das Verhalten einer Methode hat viel mit dem Begriff der mathematischen Funktion zu tun. Funktion ist ja tatsächlich auch der etwas ältere Begriff für Methode. Betrachten wir die Funktion y = sin(x). Der Name der Funktion ist sin, x ist das Argument und als Resultat liefert die Funktion den Sinus des Argumentes. Das ist der Rückgabewert. Nun gibt es auch Methoden ohne Rückgabewert, das wird dann mit dem Wort `void` gekennzeichnet.

Das Argument ist in unserem Beispiel zuerst eine fixe Zeichenkette, das ist der Text zwischen den Hochkommas. Dann ist es unsere Variable `message`, die als Argument verwendet wird. Am Bildschirm erscheint dann der aktuellen Inhalt oder Wert von `message`.

Zusammengestellt wird `message` aus drei Zeichenketten, die man einfach mit dem + Operator zusammenfügen kann.

1.2.2 Programm übersetzen, ausführen und testen

Eine IDE (*Integrated Development Environment*) unterstützt den gesamten Entwicklungsprozess von der Erstellung des Programms über die Übersetzung und Erstellung des ausführbaren Programms bis zum Testen des Programms. Die entsprechenden Befehle sind im Hauptmenü unter `Build` und `Debug` zusammengefasst. Es lohnt sich, für diese Schritte gleich von Beginn an die Tastaturkürzel zu verwenden. Die Voreinstellungen beim Visual Studio 2005 sind:

Englische Version	Deutsche Version	Tastaturkürzel
Build Solution		F6
Start Debugging		F5
Start without Debugging		Ctrl + F5
Stop Debugging		Shift + F5

Es ist unvermeidlich, dass man in einem Programm immer wieder Fehler einbaut. Man unterscheidet drei Fehlertypen.

Syntaxfehler

Da sind zunächst die Syntaxfehler. Das sind die Verstöße gegen die Rechtschreib- und Grammatikregeln der Programmiersprache. Diese Fehler entdeckt der Compiler und zeigt die fehlerhaften Stellen im Programm und in einer Fehlerliste an.

Im Beispiel sind zwei typische Fehler enthalten.

Fehler Nr. 1: `username` ist keine definierte Variable. Wie bereits erwähnt, muss man alle verwendeten Variablen zuerst definieren (vereinbaren). C# unterscheidet wie alle Sprachen der C-Familie zwischen Groß- und Kleinschreibung. Vereinbart wurde `userName`, verwendet wird dann `username`.

Fehler Nr. 2: `Console` enthält keine Methode mit dem Namen `Writeline`. Gemeint war `WriteLine`, und da beginnt Line eben wieder mit einem Großbuchstaben.

Mit einem Doppelten Mausklick auf den Fehler in der Fehlerliste gelangt man zur Stelle des Fehlers im Programm.

```
static void Main(string[] args)
{
  string userName;  // to store the users name
  string message;   // output message

  Console.WriteLine("Hello, what is your name ?");
  username = Console.ReadLine();
  message = "Hello " + userName +
    ", nice to meet you! Have fun with C#!";
  Console.Writeline(message);
```

Error List

2 Errors | 0 Warnings | 0 Messages

	Description	File	Line	Column
1	The name 'username' does not exist in the current context	Program.cs	17	4
2	'System.Console' does not contain a definition for 'Writeline'	Program.cs	20	12

Hinweis: Sicher ist Ihnen schon aufgefallen, dass schon der Editor viele Fehler anzeigt.

Sie sollten erst dann versuchen, das Programm zu starten, wenn es keine Fehler mehr enthält. In der Regel sollte man auch die Warnungen ernst nehmen, doch dazu später.

Laufzeitfehler

Das sind Fehler, die zur Laufzeit des Programms auftauchen. Das Programm gerät in einen Ausnahmezustand. Die Folgen sind unterschiedlich. Im Debug-Modus wird das Programm an der fehlerhaften Stelle angehalten und eine Fehlermeldung erscheint. Im fertigen Programm führen solche Situationen zum Absturz des Programms. Sie kenne das vermutlich zur Genüge.

Logische Fehler

Das sind Programmierfehler, die bewirken, dass ein Programm nicht das macht, macht was erwartet wird, z.B. einfach falsche Resultate produziert.

Unser fehlerfreies Programm nach dem Start:

```
file:///M:/ze/AInf/Projects/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE
Hello, what is your name ?
Karlheinz
Hello Karlheinz, nice to meet you! Have fun with C#!
Press <Enter> to end this program and close the window!
```

Konsolapplikationen verwenden als Schnittstelle zum Benutzer ein Konsolenfenster wie man es auch über Start → Ausführen →`cmd` bekommt. Manche nennen das immer noch das DOS-Fenster. Mit `cmd` startet man `cmd.exe`, eine Konsolenapplikation, die eine alternative Schnittstelle zum Betriebssystem des Computers ist.

Damit kommen wir zu den letzten beiden Zeilen des Programms:

```
Console.WriteLine(
"Press <Enter> to terminate this program!");
Console.ReadLine();
```

Im Debug-Modus (F5) verschwindet das Fenster am Ende des Programms wieder. Um die eigentlichen Resultate des Programms noch zu sehen, warten wir am Ende mit `Console.ReadLine()` auf das Drücken der Eingabe-Taste. Auf diese Situation machen wir den Benutzer aber aufmerksam.

1.3 Rechnen mit Zahlen

Wie in der Mathematik müssen wir zwischen verschiedenen Zahlentypen unterscheiden. Wichtig ist vor allem die Unterscheidung zwischen ganzen Zahlen und reellen Zahlen. Der für ganze Zahlen gedachte Datentyp ist in C# der mit dem Schlüsselwort `int` bezeichnete Typ. Für reelle Zahlen ist der Typ `double` die erste Wahl.

1.3.1 Grundrechnungsarten mit ganzen Zahlen

```
class Program
{
   static void Main(string[] args)
   {
     // integer numbers (ganze Zahlen)
     int a = 9, b;
     int c;

     b = 4;
     c = a + b;
     Console.WriteLine("{0} + {1} = {2}", a, b, c);
     c = a - b;
     Console.WriteLine("{0} - {1} = {2}", a, b, c);
     c = a * b;
     Console.WriteLine("{0} * {1} = {2}", a, b, c);
     c = a / b;
     Console.WriteLine("{0} / {1} = {2}", a, b, c);
     c = a % b;    // rest , modulus
     Console.WriteLine("{0} % {1} = {2}", a, b, c);

     Console.WriteLine("int.MinValue = {0}", int.MinValue);
     Console.WriteLine("int.MaxValue = {0}", int.MaxValue);

     Console.WriteLine
       ("\n\nPress <Enter> to terminate this program!");
     Console.ReadLine();
   }
}
```

Das Programm berechnet aus den Werten a und b den Wert c. Für a, b und c müssen Variablen des Typs int vereinbart werden. Das Programm zeigt verschiedene Varianten. Man kann nach dem Namen des Datentyps durch Bestrich getrennt, mehrere Variablen vereinbaren. Man kann schon bei der Vereinbarung der Variable einen Anfangswert geben (Initialisierung). Obwohl Zahlen automatisch mit dem Wert 0 initialisiert sind, reagiert der Compiler mit einer Warnung, wenn den Daten (Operanden) auf der rechten Seite einer Zuweisung noch keine Werte zugewiesen wurden.

Eine Anweisung

c = a *operator* b;

wird folgendermaßen umgesetzt: Die aktuellen Werte von a und b werden aus dem Speicher in zwei Register der CPU geladen. Das erfordert für a und b je einen Lesezugriff im Speicher. Die Namen a und b stehen ja schlussendlich für zwei bestimmte Speicheradressen. Auf Grund

des Datentyps ist bekannt, wie die Zahlen im Speicher codiert sind. Der Typ `int` benötigt im Speicher 4 Byte und ist in der so genannten Zweierkomplementdarstellung binär kodiert.

Dann wird c entsprechend dem Operator berechnet und das Resultat bei c abgespeichert.

Neben den Grundrechnungsarten +, -, * und / gibt es für ganze Zahlen noch den % Operator, diese Operation liefert den Rest der Ganzzahldivision. Achtung: Das Resultat der Division ist auch nur der Ganzzahlteil der Division.

Mit der Anweisung

```
Console.WriteLine("{0} * {1} = {2}", a, b, c);
```

verwenden wir ein Möglichkeit, Ausgabetext zusammenzustellen, die zunächst etwas umständlich ausschaut, auf die Dauer aber das einheitlichste und flexibelste Konzept ist.

Die Parameterliste der Methode `WriteLine` beginnt mit einer Zeichenkette, dem Format-String. Der Inhalt dieser Zeichenkette wird ausgegeben. Die Teile mit den geschweiften Klammern {0}, {1}, {2} stehen jedoch als Platzhalter für die nachfolgenden Argument. Dabei bezieht sich die Nummern zwischen den Klammern auf die durchnummerierten folgenden Argumente a, b und c. Allerdings hat das erste Argument die Nummer 0, das zweite Argument die Nummer 1 und das dritte Argument die Nummer 2. An diese Nummerierung sollte man sich rasch gewöhnen, das ist in C#, aber auch in C und C++ üblich.

Bei genauerem Hinsehen sind in C# auch die einfachen Datentypen Objekte und man kann das Schlüsselwort `int` auch als Synonym für den Klassenname Int32 verwenden. Der Editor zeigt uns nach `int`. die Verfügbaren Eigenschaften und Methoden dieser Klasse an. Die beiden Eigenschaften `MinValue` und `MaxValue` liefern zusammen den Zahlenbereich des Datentyps `int`.

1.3.2 Grundrechnungsarten mit reellen Zahlen

Dazu müssen Sie im obigen Programm nur die Variablen a, b und c nicht als int-Typen sondern als Typ `double` vereinbaren.

```
double a, b;
double c;

a = 3.1415;
b = 4E2;
```

Die Anweisung `c = a % b` ist jetzt nicht mehr gültig.

Die Zeilen für die Ausgabe liefern jetzt keine schönen Ausgaben mehr. Aber nur deshalb, weil wir die Möglichkeiten des Format-Strings nicht nutzen. Eine Verbesserung ist:

```
Console.WriteLine("{0:f3} + {1:f3} = {2:f3}", a, b, c);
```

f3 steht jetzt für: Gib die Zahlen im Fixkommaformat (f für Fix) mit drei Nachkommastellen aus.

Allgemeine Form eines Platzhalters: **{***index*[**,***alignment*][**:***formatString*]**}**

Wir stoßen hier erstmals auf eine bestimmte Syntaxschreibweise. Kursive gesetzte Namen sind durch entsprechende Angaben zu ersetzen, optionale Angaben sind in eckige Klammern [] eingeschlossen.

Auszug aus der Original-Dokumentation:

The syntax of a format item is **{***index*[**,***alignment*][**:***formatString*]**}**, which specifies a mandatory index, the optional length and alignment of the formatted text, and an optional string of format specifier characters that govern how the value of the corresponding object is formatted. The components of a format item are:

index

A zero-based integer that indicates which element in a list of objects to format. If the object specified by *index* is a null reference (**Nothing** in Visual Basic), then the format item is replaced by the empty string ("").

alignment

An optional integer indicating the minimum width of the region to contain the formatted value. If the length of the formatted value is less than *alignment*, then the region is padded with spaces. If *alignment* is negative, the formatted value is left justified in the region; if *alignment* is positive, the formatted value is right justified. If *alignment* is not specified, the length of the region is the length of the formatted value. The comma is required if *alignment* is specified.

formatString

An optional string of format specifiers. ... The colon is required if *formatString* is specified.

The leading and trailing brace characters, '{' and '}', are required. To specify a single literal brace character in format, specify two leading or trailing brace characters; that is, "{{" or "}}".

Format *Specifiers*:

Specifier	Name	Beschreibung
D oder d	Decimal	Nur für Ganzzahltypen. Die Genauigkeit definiert die Mindestanzahl von Ziffern im Ausgabetext.
X oder x	Hexadezimal	Nur für Ganzzahltypen. Ausgabe im hexadezimalen Zahlensystem mit den Ziffern 0, 1, ... 9, a, b, c, d, e, f.
F oder f	Fix-Point	Ausgabe erfolgt in der Form "-ddd.ddd..."
E oder e	Scientific Expontial	Ausgabe erfolgt in der Form "-d.ddd...E+ddd"
G oder g	General	Die Ausgabe erfolgt in der kompaktesten Form, abhängig von Datentyp and von der angegebenen Genauigkeit. Geeignet für alle numerischen Datentypen.
N oder n	Number	Ausgabe erfolgt in der Form "-d.ddd,ddd..." also mit Tausendertrennzeichen und Dezimaltrennzeichen.
C or c	Currency	Für Geldbeträge (Währungen)
P oder p	Percent	Ausgabe erfolgt in Prozent.
R oder r	Roundtrip	Garantiert, dass der Ausgabetext beim Lesen wieder in dieselbe Zahl umgewandelt werden kann.

Eine genaue Beschreibung finden Sie in der Originaldokumentation.

1.4 Einlesen von Zahlen

Ein nahe liegender Wunsch ist, die Zahlenwerte für a und b im Dialog einzulesen. Die Kommunikation mit dem Benutzer erfolgt grundsätzlich über Zeichenketten. Um Eingaben von der Konsole zu lesen, gibt es die Methode Console.ReadLine. Diese Methode liefert in einer Zeichenkette alle Eingaben in dem Moment, in dem die Eingabe mit der Eingabetaste abgeschlossen wird. Ich empfehle folgenden Programmcode:

```
static void Main(string[] args)
{
   double a, b, c;
   string input;

   Console.WriteLine("Calculate a + b");

   // Example code to read a value from the console
   // 1) Display an information
   Console.Write("a = ");
   // 2) Read the input as a string (ReadLine returns a
string)
   input = Console.ReadLine();
   // 3) Convert the input string to the desired numerical
value
   a = double.Parse(input);

   Console.Write("b = ");
   input = Console.ReadLine();
   b = double.Parse(input);

   c = a / b;
   Console.WriteLine("{0:f4} / {1:f4} = {2:g4}", a, b, c);

   ...
}
```

Die Parse Methode gibt es für jeden Datentyp.

Beachte jedoch: Die Parse-Methode führt zu einem Laufzeitfehler, wenn die Zeichenkette nicht in eine Zahl umgewandelt werden kann. Die Zeichenkette muss also einer gültigen Zahl im Dezimalsystem entsprechen. Die Einstellungen für das Dezimaltrennzeichen holt sich die Methode aus den Ländereinstellungen ihres Benutzeraccounts. Das ist also nur ein vorläufiges Konzept, um Zahlenwerte einzulesen.

1.5 Datentypen

Computer verarbeiten hauptsächlich Zahlen, Zeichen, Text und Bilder. Intern werden alle Informationen durch eine Folge von Bits dargestellt. Als Bit bezeichnet man die elementare Informationseinheit (0 oder 1). Die Bits werden in Gruppen (Bytes) eingeteilt. Dass ein Byte 8 Bits hat, ist üblich, aber nicht zwingend vorgeschrieben. Die neue, eindeutige Bezeichnung für 8 Bits ist Oktett.

Die Menge der reellen Zahlen im mathematischen Sinn kann nicht dargestellt werden. Man spricht daher besser von Gleitkommazahlen, das sind Zahlen mit Vorkomma- und Nachkommastellen, die auch einen Exponenten enthalten können. Als Datentyp für Gleitkommazahlen verwendet man hauptsächlich den Typ `double`. Der zulässige Wertebereich für den Betrag der Zahl ist 10^{-324} bis 10^{+308}, die Genauigkeit der Darstellung ist ca. 15 Dezimalstellen. Für die Speicherung benötigt ein `double`-Wert 8 Byte. Im Programmtext vorkommende Zahlenwerte, die einen Dezimalpunkt oder einen Exponenten enthalten sind vom Typ `double`. Zusätzlich gibt es einen Typ `decimal`, ein 128-bit Typ der speziell für Rechnungen mit Geldbeträgen ausgelegt ist.

Der Standardtyp für ganze Zahlen ist der Typ `int`. Der zulässige Zahlenbereich ist ungefähr ± 2 Mrd. Um diesen Zahlenbereich zu codieren, benötigt man für eine Zahl 4 Oktetts, was den heute üblichen 32-Bit Prozessoren entspricht.

Der Typ `char` ist in Unicode codiert und benötigt 16 Bit, was die Codierung von 65536 verschiedenen Zeichen ermöglicht.

Eine Besonderheit ist der Typ `bool`, der nur die Werte `true` oder `false` annehmen kann.

Tabelle der wichtigsten einfachen Datentypen:

Beschreibung	Keyword	System Type Name	Bereich	Größe
Wahrheitswert	`bool`	`System.Boolean`	false, true	
ganze Zahlen	`int`	`System.Int32`	-2 147 483 648 bis +2 147 483647	32 Bit
Gleitkomma-zahlen	`double`	`System.Double`	$\pm 5.0 \times 10^{-324}$ bis $\pm 1.7 \times 10^{308}$ Genauigkeit15-16 Stellen	64 Bit
Zeichen	`char`	`System.Char`	0000 bis FFFF	16-Bit Unicode Zeichen
Zeichenkette	`string`	`System.String`		Folge von Zeichen (Unicode)

Unicode ist der Nachfolger für den lange verwendeten 8-Bit Zeichencode. Ein Zeichencode definiert für jedes Zeichen aus einer Menge von Zeichen einen Zahlenwert. Mit 8 Bit kann man nur 256 Zeichen codieren.

Die Zeichenkette zählt eigentlich nicht mehr zu den einfachen Datentypen.

In Wirklichkeit sind auch die einfachen Datentypen in C# so implementiert, das es dazu noch viel zu sagen oder zu lernen gäbe. Das erkennt man, wenn man sieht, dass der Typname in beiden Schreibweisen gefolgt von einem Punkt in der IDE eine Liste von Eigenschaften anbietet. Das deutet darauf hin, dass auch hinter den einfachen Typen das Konzept der Objekte steckt. Wichtig ist für uns zunächst aber nur, wie man mit den einfachen Typen arbeitet.

1.5.1 Wert- und Referenz Typen

Ein weiteres wichtiges Konzept soll hier ein erstes Mal kurz angesprochen werden. Mit der Vereinbarung einer Variable erzeugt man sich ein Symbol (den Bezeichner der Variable), das für unterschiedliche Dinge steht.

Für Wertetypen (*value types*) enthält die Variable (die zugehörigen Speicherzellen) den Wert selber.

```
int a = 2;
int b = 3;
int c;
c = a * b;
```

In diesem Ausdruck steht a für den Wert 2 und b für den Wert 3, d.h. es wird 2 * 3 berechnet. Das ist vermutlich auch das, was allgemein erwartet wird.

Während der Ausführung eines Programms sind die Daten (und auch der Code) eines Programms irgendwo im Speicher des Computers abgelegt. Wird eine Methode aufgerufen, so werden zunächst alle Variablen der Methode in einem Speicherbereich abgelegt, den man als Stack bezeichnet. Für unser Beispiel könnte das bildlich so dargestellt werden:

Variable	Inhalt	Speicheradresse
c	6	…2008
b	3	…2004
a	2	…2000

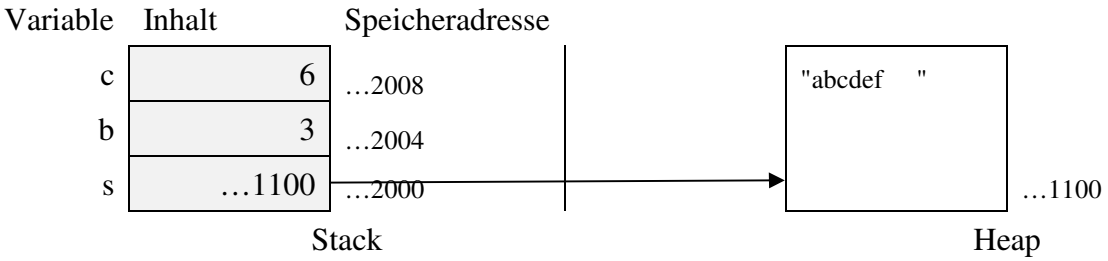
Am Ende der Ausführung einer Methode wird der von den Variablen belegte Speicherplatz wieder frei gegeben.

Wenn sie sich für jede Variable ein Platt Papier vorstellen, funktioniert ein Stack so:

Für jede neue Variable wird ein Blatt Papier oben auf den Stoß gelegt. Am Ende werden die Blätter in der umgekehrten Reihenfolge wieder entfernt. Ein Stack funktioniert nach dem FILO (*First In, Last Out*) Prinzip.

Die Datentypen `int`, `double`, `char`, `bool` sind Wertetypen.

Die zweite Kategorie von Typen in C# sind die Referenztypen (*reference types*). Zu dieser Kategorie zählen alle Objekte. Der Typ `string` ist ein Beispiel für einen Referenztyp. Enthält eine Methode z.B. zwei int-Werte a und b auch eine Zeichenkette s, so ist die Situation am Speicher so:



Für Referenztypen enthält die Variable eine Referenz (also eine Speicheradresse) auf den Beginn des Speicherbereichs, der für dieses Objekt verwendet wird. Auf den heute üblichen Systemen ist eine Speicheradresse 4 Bytes lang.

Ein wichtiger Unterschied ergibt sich bei der einfachen Zuweisung einer Variablen an eine andere Variable.

```
int a = 4;
int b;
b = a;
ObjectTyp o1, o2;
o1 = new ObjectType();
o2 = o1;
```

Nach b = a enthält b eine Kopie von a. Eine spätere Änderung von b betrifft a nicht.

Für obj1 und obj2 sind je 4 Byte am Stack reserviert. Am Heap benötigt das Objekt wesentlich mehr Speicher.

```
new ObjectType()
```

erzeugt ein Objekt und speichert es am Heap, die Adresse am Heap wird zum Wert der Variablen obj1, obj1 enthält also eine Referenz auf das Objekt.

`o2 = o1` kopiert nicht das ganze Objekt, sondern nur die Referenz auf das Objekt. o1 und o2 "zeigen" auf dasselbe Objekt.

1.6 Grundbegriffe der Syntax

Für eine Programmiersprache sind genaue Regeln definiert, die festlegen, wie der Text eines korrekten Programms formuliert sein muss. Die Syntax einer Programmiersprache definiert, ausgehend von den erlaubten Zeichen, wie man aus diesen Zeichen Grundsymbole und "Sätze" bildet.

1.6.1 Zeichensatz und Wörter

C# verwendet als Zeichensatz für den Quellcode eines Programms den Unicode-Zeichensatz. Gegenüber den Regeln in C und C++ ergeben sich daraus Regeln und Möglichkeiten, deren Beschreibung Kenntnisse erfordert, die man in diesem Rahmen sonst nicht benötigt. Jedenfalls ist eine Einteilung in Buchstaben (*letters*), Ziffern(*digits*), Sonderzeichen und Zeilentrennzeichen (*line terminators*) und Zwischenraumzeichen (*white space*) sinnvoll.

Buchstaben	a b c ... z A B C ... Z
Ziffern	0 1 2 3 4 5 6 7 8 9

Sonderzeichen	! " # % & ' ( ) * + , - . / …
Zeilentrennzeichen	CR (*Carriage return*), LF (*Line feed*), CR+LF, zwei weitere Unicodezeichen
Zwischenraumzeichen	Leerzeichen (*space*), HT (*horizontal tab*), VT (*vertical tab*), FF (*form feed*)

Aus dem Zeichenstrom des Programms bildet der Compiler zusammengehörige Einheiten (*token*). Der Compiler erkennt diese zusammengehörigen Einheiten dann als:

- Schlüsselwörter (*keyword*)
- Bezeichner (*identifier*), sind vom Programmierer gewählte Namen
- Konstanten (*literals*)
- Operatoren und Satzzeichen (*punctuators*)
- Spezielle Unicode Sequenzen

Der Compiler kümmert sich nicht um die Zeilenstruktur des Quelltextes. Er kümmert sich auch nicht darum, ob statt eines Leerzeichens mehrere Leerzeichen als Trennzeichen verwendet werden. Für den Compiler ist ein Programm ein eindimensionaler Zeichenstrom, für den Leser ist er ein zweidimensionales Gebilde, dessen Form (*Layout*) die Lesbarkeit des Programmcodes entscheidend beeinflusst.

1.6.2 Syntaxnotation

Die Syntax einer Programmiersprache muss irgendwie formal dargestellt werden. Ein Beispiel für eine solche Darstellung von Regeln ist die Online-Hilfe des Betriebssystems, wo z.B. angegeben wird, wie man eine Kommandozeile zu formulieren hat.

Die Grammatik der C Sprachenfamilie wir in einer Form dargestellt, die auch von einem Programm einfach gelesen werden. Diese Regeln definieren die einzelnen Bausteine (man könnte Sagen Sätze) der Sprache und beschrieben wie ein Satz gebildet wird. Das Original (*C# Language Specification*) spricht von *grammar production*. Die Definition eines Syntaxbegriffs verweist auf andere Syntaxbegriffe und auf Symbole und Zeichen, die fixe, endgültige Symbole (*terminal symbols*) der Sprache sind.

Ein Syntaxbegriff wird *kursiv* geschrieben und mit einem Doppelpunkt abgeschlossen. Der Doppelpunkt kann gelesen werden als "ist definiert wie folgt".

Beispiel:

```
token:
  identifier
  keyword
  literal
  operator-or-punctuator
```

Das bedeutet ein *token* ist entweder ein *identifier* oder ein *keyword* oder ein *literal* oder ein *operator-or-punctuator*. Eine Auswahl, aus der ein Element gewählt werden muss, wird also untereinander geschrieben. Statt jede Alternative in eine eigene Zeile zu schreiben, kann man auch schreiben:

```
decimal-digit: one of
    0 1 2 3 4 5 6 7 8 9
```

Ein tief gestelltes $_{opt}$ wird verwendet, um anzuzeigen, dass dieses Symbol optional ist. Endgültige Symbole sind nicht kursiv in einer Schrift mit fixer Zeichenbreite angeschrieben. Zur besseren Hervorhebung verwende ich noch Fettschrift.

Es ist nicht Sinn einer Einführung, die Grammatik von C# lückenlos darzustellen. Wenn aber wichtige Regeln definiert werden, dann macht es Sinn, dafür eine einheitliche Notation zu verwenden.

1.6.3 Schlüsselwörter

Schlüsselwörter sind reservierte Wörter, welche als Bezeichner nicht verwendet werden können.

keyword: one of

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			

1.7 Ausdrücke (*expressions*) und Operatoren (*operators*)

Mathematische Terme formuliert auch der Anfänger intuitiv zumindest syntaktisch richtig:

```
u = r*i;
p = u*i;
s = s0 + v*t;
s = g*t*t/2;
```

Eine andere Schreibweise der letzten Formel, z.B. in der Form

```
s = 1/2*g*t*t
```

erzeugt allerdings ein nicht beabsichtigtes Resultat. 1/2 hat in C, C++ und C# das Resultat 0, wie wir gleich sehen werden. Für die Verknüpfung von Ganzzahlwerten und/oder Gleitkommawerten stehen folgende arithmetische Operatoren zur Verfügung:

arithmetic-operator: one of **+ - * / %**

`+` , `-` , `*` , `/` sind die Operatoren für die Grundrechnungsarten. Wenn ein Ganzzahlwert mit einem Gleitkommawert verknüpft wird, so ist das Resultat ein Gleitkommawert. Sind beide Operanden vom gleichen Typ, so hat auch das Resultat diesen Typ. Ein häufiger Fehler ist, nicht zu beachten, daß die Division von zwei `int`-Werten wieder einen int-Wert, also eine ganze Zahl ergibt, man nennt das eine Ganzzahldivision:

```
8 / 2 = 4      8 / 3 = 2      1 / 2 = 0
```

Als Ergänzung zur Ganzzahldivision ergibt der `%` -Operator den Rest der Ganzzahldivision. Die beiden Operanden müssen `int`-Werte sein.

```
8 % 2 = 0      8 % 3 = 2      1 % 2 = 1
```

Arithmetische Ausdrücke genügen der Syntaxregel

arithmetic-expression: *expr arithmetic-operator expr*

Auch die Zuweisung zählt zur Syntaxkategorie Ausdruck:

assignment-expression: *lvalue assignment-operator expr*

lvalue ::= *identifier* ...

```
a = 4
```

ist ein solcher Zuweisungsausdruck. Als *lvalue* wird der Bezeichner `a` verwendet; = ist der wichtigste Zuweisungsoperator (*assignment-operator*); die Konstante `4` ist ein besonders einfacher Ausdruck. Ein wichtiges Detail der Syntax - das wir später noch besprechen - ist, wie aus einem Ausdruck eine Anweisung entsteht:

expression-statement ::= *expression*$_{opt}$;

Die arithmetischen Operatoren verknüpfen immer zwei Ausdrücke miteinander, welche als Operanden wirken. Plus- und Minuszeichen treten auch als unäre Operatoren auf, wenn man sie einfach als Vorzeichen vor einen Ausdruck setzt. Unäre Operatoren wirken nur auf einen Operanden.

Zwei weitere Gruppen von Ausdrücken mit zugehörigen Operatoren werden durch folgende Syntaxregeln definiert:

```
equality-expression   ::=   expression == expression
                            expression != expression

relational-expression ::=   expression <  expression
                            expression <= expression
                            expression >  expression
                            expression >= expression
```

Beispiele für solche Ausdrücke sind

```
a == b      a != 5      b  <  5       x  >=  3.1415
```

Diese Ausdrücke prüfen, ob eine Relation richtig (wahr) oder falsch ist, das Resultat ist also ein Wahrheitswert. Ist der Ausdruck wahr, so ist das Resultat true, sonst false.

Mit den Vereinbarungen

```
int a, b;
double x;
```

und den Zuweisungen

```
a = 4; b = 6; x = 4.0;
```

ergeben sich für folgende Ausdrücke die Resultate

```
a == b          false
a != 4          false
4 <  5          true
x >= 3.1415     true
```

Logische Ausdrücke verknüpfen Wahrheitswerte mit den logischen Operatoren NICHT, UND, ODER (*NOT, AND, OR*):

```
logical-expression:
   ! expression
   expression || expression
   expression && expression
```

C# verwendet für diese Operatoren die Sonderzeichen

! für NOT, dieser Operator wirkt nur auf einen Operanden, es handelt sich um einen unären Operator mit folgender Wirkung:

x	!x
true	false
false	true

&& für das logische UND, zwei logische Ausdrücke werden nach folgenden Regeln verknüpft (Wahrheitstabelle):

a	b	a && b
false	false	false
false	true	false
true	false	false
true	true	true

|| ist das Symbol für das logische ODER. Es gelten die Regeln:

a	b	a || b
false	false	false
false	true	true
true	false	true
true	true	true

Typische Beispiele dazu:

`( c >= 'a' ) && ( c <= 'z' )`	Prüft, ob ein Zeichen ein Kleinbuchstabe ist.
`( x > 0 ) && ( y > 0 )`	Prüft, ob sowohl die x-Koordinate als auch die y-Koordinate positiv ist.

Ausdrücke mit diesen Operatoren werden nur ausgewertet, solange das Resultat noch nicht feststeht. Im Ausdruck `x && y` wird der Ausdruck y nur berechnet, wenn x `true` ist. Die

Operatoren && und || werden daher auch *conditional logical operators* oder *short-circuiting operators* genannt.

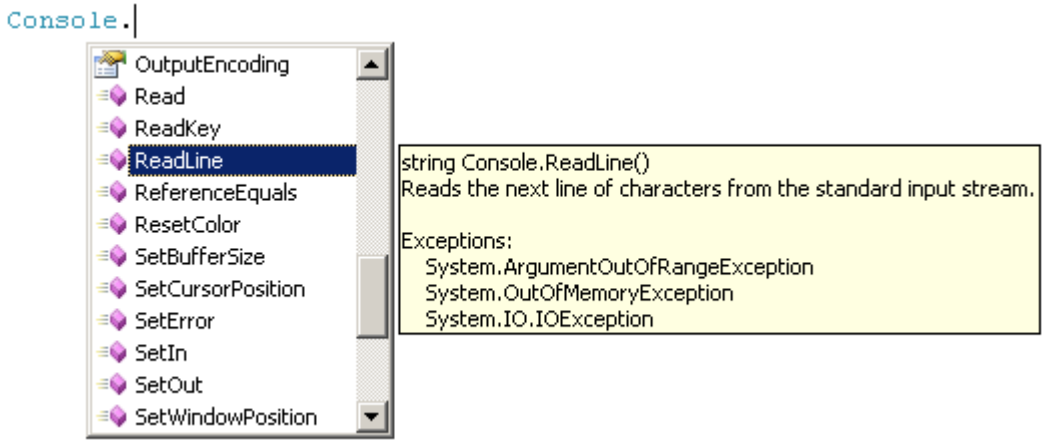
Für das Zusammentreffen von mehreren Operatoren in einem Ausdruck muss definiert sein, welche Vorrangregeln gelten. Natürlich gilt z.B. die Regel "Punktrechnung vor Strichrechnung". Treffen Operatoren gleicher Priorität zusammen, so ist festgelegt, in welcher Richtung ausgewertet (zusammengefasst) wird. Die folgende Tabelle zeigt die Rangordnung der uns bereits bekannten Operatoren.

Operatoren	Auswertungsrichtung	
( )	links nach rechts	hohe Priorität
! + - (unär)	rechts nach links	
* / %	links nach rechts	
+ -	links nach rechts	
< <= > >=	links nach rechts	
== !=	links nach rechts	
&&	links nach rechts	
||	links nach rechts	
=	rechts nach links	niedere Priorität

Im Zweifelsfall, aber auch zur besseren Lesbarkeit, setzen wir Klammern.

1.8 Frameworkobjekte verwenden

In jedem noch so kleinen Programm verwendet man Funktionalität aus dem Framework und ist damit mit den Konzepten der objektorientierten Programmierung konfrontiert.



Der Editor liefert mit seiner "*Intellisense*" Funktion zu jeder Frameworkklasse oder zu jedem Frameworkobjekt eine Liste der verfügbaren Eigenschaften, Methoden. Nach der Auswahl einer Eigenschaft oder Methode wird auch ein Hilfetext angezeigt.

Eigenschaften:

Mit den Eigenschaften (Papier + Hand Symbol) kann man bestimmte Datenwerte abfragen und setzen. So wie Datenwerte einen Datentyp haben hat auch eine Eigenschaft einen Datentyp. Eine typische Hilfe zu einer Eigenschaft ist z.B. für die Eigenschaft `CursorLeft`:

Gets or sets the column position of the cursor within the buffer area.

`Get` ist der Lesezugriff auf die Eigenschaft. `Set` ist der Schreibzugriff auf die Eigenschaft. Welcher Zugriff verwendet wird, ergibt sich aus dem Kontext. Verwendet man die Eigenschaft auf der linken Seite einer Zuweisung, so wird "Set" verwendet, sonst "Get". Beachte, dass Set nicht immer zur Verfügung steht.

Methoden:

Studiert man die Namen der `Console` Methoden etwas, so fallen zwei Gruppen auf. Oft ist der Methodenname ein Verb oder ein "Tun-Wort" wie man in der Grundschule lernt. Das heißt, es wird etwas getan, etwas ausgeführt. Beispiele für solche Methoden sind: Beep, Clear, `Read`, `ReadLine`, `Write`, `WriteLine` etc. Die Methoden der zweiten Gruppe beginnen oft mit Set oder Reset. Diese Methoden verändern meist mehrere Datenwerte und damit Eigenschaften des `Console` Objektes. Im Gegensatz zu Eigenschaften haben Methoden eine Parameterliste. Über diese Liste kann die Methode mit Parametern (Argumenten) versorgt werden. Die Hilfe zeigt immer an, welche Argumente abgegeben werden müssen.

1.9 Programmierstil und Namensgebung

Die Welt der professionellen Programmierer hält sich an einen bestimmten Programmierstil und damit an vereinbarte Konventionen. Jedes Programmierteam hat solche Regeln vereinbart und die einzelnen Teammitglieder müssen sich an diese Regeln halten. Das Vorbild für C# sind Regeln, wie sie von den Entwicklern der Sprache C# und des .NET Frameworks vorgeschlagen werden und einzelne Buch Klassiker zu diesem Thema. Die Bibel zu diesem Thema ist wohl: "Framework Design Guidlines" von Krzysztof Cwalina und Brad Abrams.

1.9.1 Allgemeine Regeln und Begriffe:

Ist ein Name ein zusammengesetztes Wort, so beginnen die einzelnen Wörter mit einem Großbuchstaben.
`maxLength, maxNumber, minNumber`

Diese Schreibweise eines zusammengesetzten Wortes, erster Buchstabe klein, die weiterer ersten Buchstaben eines Wortes wieder groß, nennt man "***c**amel**C**asing*".

Ist auch der erste Buchstabe groß, so spricht man von "***P**ascal**C**asing*".

1.9.2 Namenskonventionen (*naming conventions*)

Verwende *camelCasing* für die Namen von Variablen und Argumenten einer Methode.
`length, count, number, text`

Verwende PascalCasing für Namensräume, Klassen und andere Datentypen und für Methoden und Eigenschaften.

Verwende die C# Aliases anstatt der Frameworknamen.
`int` statt `Int32`, `string` statt `String` ...

Verwende aussagekräftige Namen
`konto` statt `k`, `customerName` statt `cn oder kn`

Eine Ausnahme sind eventuell die üblichen physikalischen Größen wie v, s, t, F, W, P etc. wobei in dem Zusammenhang auch Probleme mit Groß- / Kleinbuchstabe am Beginn des Namens auftauchen.

Eine weitere Ausnahme sind die typischen Bezeichner i, j, k, l, m, n wie sie in der Mathematik als Indizes, Indexgrenzen bei indizierten Namen, Summen etc. verwendet werden. Diese Bezeichner verwendet auch der Programmierer oft als Kontrollvariable in Schleifen und als Index. Sie verwirren einen erfahrenen Programmierer, wenn Sie diese Buchstaben z.B. für Daten des Typs `double` oder `string` verwenden.

Verwende im Code selber nach Möglichkeit Englisch als Sprache für die Namen und Kommentare. In einem international besetzten Team, das ist heute die Regel, ist das Pflicht.

1.10 Kommentare

Die begleitende Kommentierung des Programmtextes ist wichtig. Jeder erfahrene Programmierer weiß, wie schwer man nach längerer Zeit auch selber geschriebene Programme lesen kann, wenn der Programmcode keine Kommentare enthält. Es lohnt sich, Programme von Beginn an sauber zu kommentieren.

In C# ist die übliche Art die Verwendung von "//" um eine ganze Zeile oder den Rest der Zeile als Kommentar zu deklarieren.

```
string dateString;      // current date in format dd.mm.yyy
// read the next character from the ports
```

C# erlaubt aber auch die alte C Kommentierung mit

```
/*  Beginn des Kommentars

End des Kommentars */
/*
// Damit sind auch verschachtelte Kommentare möglich!
*/
```

Was soll kommentiert werden?

* Dateiname, Autor, Datum und Zweck eines Programms am Beginn des Programmcodes
* die verwendeten Variablen
* die Funktion von einzelnen Programmabschnitten
* besondere Anweisungen

Ein Programm kann auch überkommentiert sein. Nicht kommentiert werden Anweisungen, deren Zweck ohnehin klar ist, also z.B.:

```
Console.WriteLine(i);     // print i
y = x*y - 3/x;            // calculate y
```

Für Programme in einem Lehrbuch gelten etwas andere Überlegungen. Dem Anfänger rate ich, alles zu kommentieren, was für ihn neu ist. Damit wird die Programmsammlung zum persönlichen Lehrbuch.