C#

Stand: 01.12.2013

Inhaltsverzeichnis

C#-Einstieg: Das erste C#-Programm ... 4

Ein C#-Projekt anlegen ... 4

Erstes Programm: „Hello-World“ ... 5

Datentypen/Variablen und Ein- und Ausgabe ... 7

Variablen in C# / Datentypen und Deklaration ... 7

Deklaration einer Variablen / Initialisierung einer Variablen ... 8

Ein- und Ausgabe ... 8

Ausgabe in C# ... 8

Eingabe mit ReadLine ... 9

Operatoren ... 11

Arithmetische Operatoren ... 11

Programmieraufgaben (I): Ein-und Ausgabe & Datentypen int/double und Grundrechenarten ... 13

Vergleichsoperatoren ... 14

Logische Operatoren ... 14

Verzweigung ... 16

Bedingte Anweisung (if) und Verzeigung (if-else) ... 16

Beispiel-Programm ... 17

Programmieraufgaben (II): if-else-Anweisung ... 18

Switch-Anweisung (Mehrfach Verzweigung) ... 20

Aufbau der switch-Anweisung ... 20

Vergleich zwischen if-else-Befehl und switch-case-Befehl: ... 20

Beispielprogramm ... 21

Schleifen (Wiederholungsanweisungen) ... 22

Bedingungsabhängige Schleife (while- und do-while-Schleife) ... 22

Beispiel für eine do-while-Schleife ... 23

Programmieraufgaben (III): while- und do-while-Schleife ... 24

Zählergesteuerte Schleife (for-Schleife) ... 26

Programmieraufgaben (IV): for-Schleife ... 28

Programmieraufgaben (V): Zahlenratespiel 29

Programmieraufgaben (VI):Wurzel ziehen mit Intervallhalbierungsverfahren..... 30

Programmieraufgaben (VII): Weitere Aufgaben..... 32

Programmieraufgaben (VIII): Einmaleins / Grundrechenarten..... 34

Programmieraufgaben (IX): von A.Fey und Z.Genc..... 38

Erste Windowsanwendung 52

Programmieraufgaben(IX): Taschenrechner 57

Literatur..... 58

Indexverzeichnis..... 59

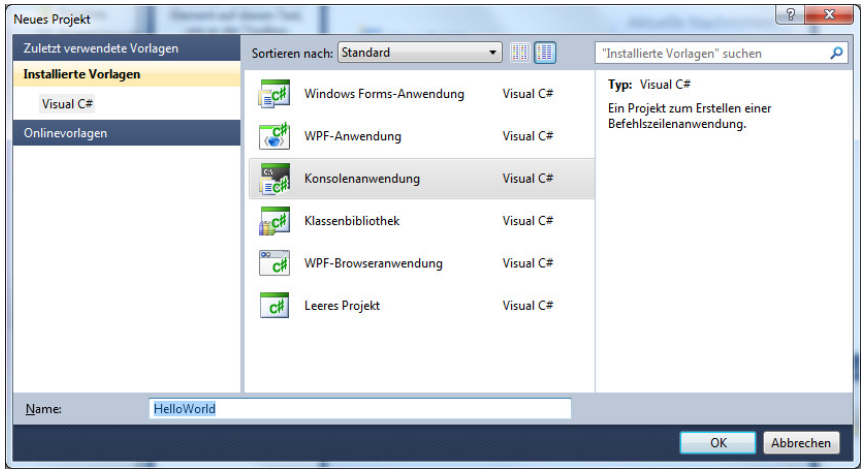
C#-Einstieg: Das erste C#-Programm

Ein C#-Projekt anlegen

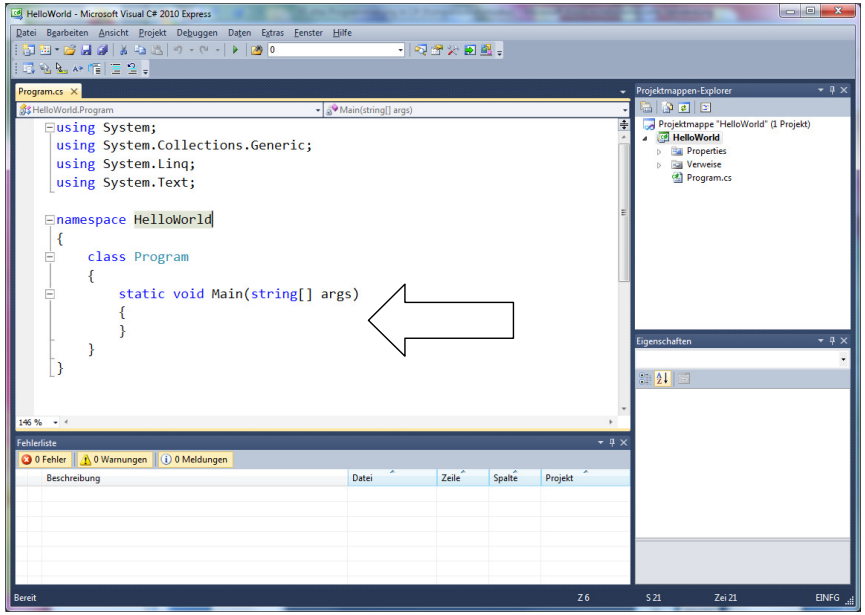
Um C#-Programme zu entwickeln, verwenden wir im Unterricht die Entwicklungsumgebung "Microsoft Visual C# 2010". Eine kostenlose Version des Programms "Microsoft Visual C# 2010 Express Edition" können Sie im Internet herunterladen.

Ein neues Projekt wird wie folgt angelegt:

- Microsoft Visual C# 2010 Express Edition starten
- Wählen Sie im Menü: Datei → Neues Projekt: Konsolenanwendung wählen
- Name für das Projekt eingeben: HelloWorld →OK



Nach dem bestätigen mit "OK" wird ein neues Projekt angelegt und im Entwicklungsumgebung angezeigt:



Wir werden uns zu Beginn auf das "Hauptprogramm" Main beschränken, dieser befindet sich in der Datei "Program.cs". Bis wir Klassen und Methoden kennenlernen werden alle C#-Anweisungen innerhalb der Main-Methode geschrieben. (Siehe Pfeil in der obigen Abbildung)

Erstes Programm: „Hello-World“

Ihr erstes Programm soll auf dem Bildschirm "Hello World" schreiben.

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

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World! ");
        }
    }
}
```


Die Anweisung "`Console.WriteLine("Hello World! ");`" schreibt auf dem Bildschirm eine Zeichenkette[1] und macht danach einen Zeilenumbruch.

Nach dem starten mit **Strg+F5** erscheint ein Fenster mit folgender Bildschirmausgabe:

[1] Eine Zeichenkette ist eine Reihe von endlich vielen Zeichen, also Buchstaben, Ziffern und Sonderzeichen, die in doppelten Anführungsstrichen eingerahmt sind, z.B. " Text 34 "

Datentypen/Variablen und Ein- und Ausgabe

Eine elementare Aufgabe von Programmen ist die Speicherung und Verarbeitung von Daten. Erst durch die Möglichkeit Werte einzulesen und zu verarbeiten beginnt das eigentliche Programmieren.

Variablen in C# / Datentypen und Deklaration

Variablen dienen dazu, Werte zu speichern. Sie sind Platzhalter für ein beliebigen Wert. In C# muss vorher festgelegt werden, welche Art von Wert eine Variable speichern soll. Man spricht dabei vom **Datentyp** der Variable. Der Datentyp bestimmt ganz genau die Art des Wertes und den Wertebereich.
Jede Variable hat ein speziellen Datentyp und kann nicht jeden Wert speichern.

Die folgende Tabelle zeigt die elementaren Datentypen und ihre Gültigkeitsbereiche, die wir im Unterricht verwenden werden:

Datentyp	Beschreibung	Größe in Byte	Wertebereich
`int`	Dieser Datentyp dient zur Speicherung von positiven und negativen ganzzahligen Werten.	4 Byte	-2147483648 bis -2147483647 (-2^{31} bis 2^{31})
`long`	Dieser Datentyp dient zur Speicherung von positiven und negativen ganzzahligen Werten.	8 Byte	-9223372036854775808 bis -9223372036854775807 (-2^{63} bis 2^{63})
`double`	Dieser Datentyp dient zur Speicherung von Gleitpunktzahlen.	8 Byte	-1,797693134862E+308 bis 1,797693134862E+308
`bool`	Mit diesem Datentyp können boolesche Werte gespeichert werden.	1 Byte	`true` oder `false`
`string`	Dieser Datentyp dient zur Speicherung von Zeichenketten.	Je nach Bedarf	Alle Zeichen aus dem Unicode[2]-Zeichensatz.

[2] **Unicode** (Aussprachen: [ˈjuːnikoːt]) ist ein internationaler Standard, in dem langfristig für jedes sinntragende Schriftzeichen oder Textelement aller bekannten Schriftkulturen und Zeichensysteme ein digitaler Code festgelegt wird. Ziel ist es, die Verwendung unterschiedlicher und inkompatibler Kodierungen in verschiedenen Ländern oder Kulturkreisen zu beseitigen.

Deklaration einer Variablen / Initialisierung einer Variablen

Variablen werden angelegt ("deklariert") mit Hilfe der elementaren Datentypen und eines Bezeichners (Variablenname) für die Variable.

Beispiele:

- eine Variable vom Typ int anlegen:
  `int zahl;`
- eine Variable vom Typ double anlegen:
  `double zahl2;`
- eine Variable vom Typ string anlegen:
  `string eingabe;`

Bei der Deklaration einer Variablen kann sofort eine Initialisierung erfolgen. Damit wird der Variablen ein Wert zugewiesen. Das geschieht mit dem so genannten Zuweisungsoperator "=".

Beispiele:

- eine Variable vom Typ int anlegen und initialisieren:
  `int zahl=30;`
- eine Variable vom Typ double anlegen und initialisieren:
  `double zahl2=3.45;`
- eine Variable vom Typ string anlegen und initialisieren:
  `string eingabe="hallo xy";`

Beachte: Vor der ersten Verwendung einer Variablen nach der Deklaration muss der Variable ein Wert zugewiesen worden sein.

Ein- und Ausgabe

Ausgabe in C#

Ausgabe von Text und Variablen

Mit dem bereits bekannten Methode `WriteLine()` können sowohl Zeichenketten als auch Inhalte von Variablen ausgegeben werden.

```
class Program
{
    static void Main(string[] args)
    {
        int zahl = 20;
        double zahl2 = 2.35;
        string eingabe = "Text xy";

        Console.WriteLine("Zeichenkette wird ausgeben");
        Console.WriteLine(zahl);
        Console.WriteLine("Eine ganze Zahl:"+zahl+" Dezimanlzahl:"+zahl2);
        Console.WriteLine("Zeichenkette:"+eingabe);
    }
}
```

Der +-Operator verknüpft die Zeichenkette mit dem Inhalt der Variablen.

Durch den +-Operator werden die Inhalte der Variablen automatisch in Zeichenketten umgewandelt und werden dann mit anderen Zeichenketten verknüpft und auf dem Bildschirm ausgegeben.

Nach dem Starten sieht die Ausgabe wie folgt aus:

Ausgabe von Sonderzeichen

Alle Sonderzeichen (Escape-Sequenzen) werden in Zeichenkette durch den Backslash "\" eingeleitet. Einige Beispiele für Escape-Sequenzen:

- `\a` gibt einen Piepston aus
- `\b` Backspace - Cursor geht einen Schritt zurück
- `\t` Standardtabulator
- `\n` new line - Zeilenumbruch
- `\"` Anführungszeichen werden ausgegeben.
- `\\` gibt einen Backslash aus.

Eingabe mit ReadLine

Zeichenketten einlesen kann mit der Methode **`ReadLine()`** der Klasse `Console` realisiert werden. Die Methode liest die Eingabe über die Tastatur und speichert ihn in einer Zeichenkette ab. Diese Zeichenkette kann dann einer Variablen zugewiesen werden.

Die Methode **`ReadLine()`** speichert die Eingabe immer in einer Zeichenkette. Deshalb muss eine Umwandung (Konvertierung) vorgenommen werden, nicht eine Zeichenkette sondern eine Zahl (int oder double) über die Tastatur eingelesen werden soll. Für die Umwandlung steht die Klasse `Convert` zu Verfügung. Diese Klasse verfügt über viele Methoden, die eine Zeichenkette in ein beliebiges anderes Format umwandeln.

Einige Konvertierungmethoden der Klasse `Convert`:

- Wandelt den Wert in der Klammer in ein Integer-Wert mit 16 Bit (2 Byte) um.
  `Convert.ToInt16(...);`
- Wandelt den Wert in der Klammer in ein Integer-Wert mit 32 Bit (4 Byte) um.
  `Convert.ToInt32(...);`
- Wandelt den Wert in der Klammer in ein Double-Wert um.
  `Convert.ToDouble(...);`
- Wandelt den Wert in der Klammer in eine Zeichenkette (String) um.
  `Convert.ToString(...);`

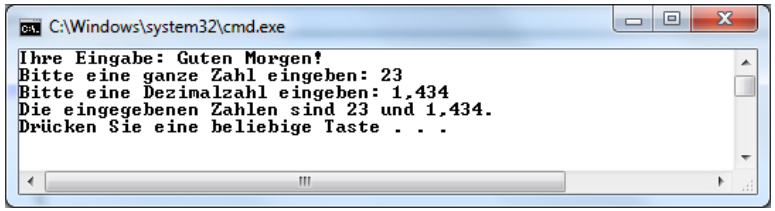
Beispiel Programm:

```
    class Program
    {
        static void Main(string[] args)
        {
            int ganzeZahl;
            double dezimalzahl;
            string eingabe;

            Console.Write("Bitte eine Eingabe machen: ");
            eingabe = Console.ReadLine();

            Console.WriteLine("Ihre Eingabe: " + eingabe);

            Console.Write("Bitte eine ganze Zahl eingeben: ");
            ganzeZahl = Convert.ToInt32(Console.ReadLine());

            Console.Write("Bitte eine Dezimalzahl eingeben: ");
            dezimalzahl = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Die eingegebenen Zahlen sind " +
ganzeZahl + " und " + dezimalzahl + ".");
        }
    }
```

Die Methode speichert die Tastatureingabe in eine Zeichenkette und diese wird dann in die string-Variablen eingabe zugewiesen.

Die Methode `ToInt32` wandelt die Zeichenkette der `ReadLine`-Methode in ein Integer-Wert mit 32Bit (4 Byte) um.

Nach dem Starten könnte die Bildschirmausgabe so aussehen:



Operatoren

Die Operatoren sind ein wichtiger Bestandteil jeder Programmiersprache auf die man nicht verzichten kann.

Arithmetische Operatoren

C# kennt die Operatoren der vier Grundrechenarten Addition, Subtraktion, Division und Multiplikation. Darüber hinaus haben diese Operatoren in anderem Kontext eine besondere Bedeutung. Die folgende Tabelle gibt einen allgemeinen Überblick.

Operator	Bedeutung
+	Hat zwei Funktionalitäten: Als **Additionsoperator** bildet er die Summe zweier Operanden (x + y). Als **Vorzeichenoperator** beschreibt er eine positive Zahl (+x).
–	Hat ebenfalls zwei Funktionalitäten: Als **Subtraktionsoperator** eingesetzt, bildet er die Differenz zweier Operanden (x - y). Als **Vorzeichenoperator** beschreibt er eine negative Zahl (-x).
*	**Multiplikationsoperator** multipliziert zwei Operanden (x * y).
/	**Divisionsoperator** dividiert zwei Operanden (x / y), behält den Nachkommateil der Division.
%	**Modulo-Operator (Restwertoperato)r**; dividiert zwei Operanden und liefert als Ergebnis den Restwert der Operation (x % y).
++	**Inkrementoperator**: Erhöht den Inhalt des Operanden um 1. Das Ergebnis der Operation ++x ist der Wert des Operanden nach der Erhöhung (Präfixinkrementoperation). Das Ergebnis der Operation x++ ist der Wert des Operanden vor der Erhöhung (Postfixinkrementoperation).
– –	**Dekrementoperator:** Verringert den Inhalt des Operanden um 1. Das Ergebnis der Operation --x ist der Wert des Operanden nach der Verringerung (Präfixdekrementoperation). Das Ergebnis der Operation x-- ist der Wert des Operanden vor der Verringerung (Postfixdekrementoperation).

Der Einsatz der Operatoren zur Formulierung mathematischer Ausdrücke ist trivial. Zwei Operanden werden miteinander verknüpft, und das Ergebnis der Operation wird der links vom Zuweisungsoperator stehenden Variablen zugewiesen.


```
int zahl1 = 30;
int zahl2 = 7;
int ergebnis;
ergebnis = zahl1 + zahl2; // Variable ergebnis hat den Wert 37
ergebnis = zahl1 - zahl2; // Variable ergebnis hat den Wert 23
ergebnis = zahl1 * zahl2; // Variable ergebnis hat den Wert 210
ergebnis = zahl1 / zahl2; // Variable ergebnis hat den Wert 4
ergebnis = zahl1 % zahl2; // Variable ergebnis hat den Wert 2
zahl1++;  // Variable zahl1 hat den Wert 31
zahl2--;  // Variable zahl2 hat den Wert 6
```

Beispielprogramm: Das folgende Programm liest zwei Zahlen von der Tastatur und addiert die beiden Zahlen und gibt dann das Ergebnis aus.

```
class Program
{
    static void Main(string[] args)
    {
        // 3 Variablen werden deklariert
        int zahl1;
        int zahl2;
        int summe;

        // Text wird ausgegeben und eine Zahl wird eingelesen
        Console.Write("1. ganze Zahl  eingeben : ");
        zahl1 = Convert.ToInt32(Console.ReadLine());

        // Text wird ausgegeben und eine Zahl wird eingelesen
        Console.Write("2. ganze Zahl  eingeben : ");
        zahl2 = Convert.ToInt32(Console.ReadLine());

        // Die Zahlen in den Variablen zahl1 und zahl2 werden
        // addiert und in die Variable summe gespeichert.
        summe = zahl1 + zahl2;

        // Die Zahlen und die Summe wird ausgegeben.
        Console.WriteLine(zahl1+ " + " +zahl2+ " = " +summe);
    }
}
```

Nach dem Starten könnte die Bildschirmausgabe so aussehen:

Programmieraufgaben (I): Ein-und Ausgabe & Datentypen int/double und Grundrechenarten

1.Aufgabe **Quadratzahlen**

- Eingabe: eine Zahl
- Ausgabe: Quadrat dieser Zahl
- z.B.:

```
zahl eingeben = 5
5*5 = 25
```

2.Aufgabe **Ohmsches Gesetz**

- Eingabe: Spannung U und Stromstärke I
- Ausgabe: Widerstand R[3]
- z.B.:

```
Spannung (V) eingeben = 50
Stromstärke (A) eingeben = 10
Widerstand =  5 Ohm
```

3.Aufgabe **Grundrechenarten**

- Eingabe: zwei Zahl (**zahl1** und **zahl2**)
- Ausgabe:
  - Summe: **zahl1+zahl2**;
  - Differenz: **zahl1-zahl2**;
  - Produkt: **zahl1*zahl2**;
  - Quotient: **zahl1/zahl2**
- Die Variablen **zahl1** und **zahl2** zuerst mit dem Datentyp **int** später mit **double** deklarieren.
- z.B.:

```
1.zahl eingeben = 50
2.zahl eingeben = 10
50 + 10 = 60
50 - 10 = 40
50 * 10 = 500
50 / 10 = 5
```

[3] R = U/I

Vergleichsoperatoren

Mit Vergleichsoperatoren kann man zwei Werte vergleichen und das Ergebnis ist ein boolescher Wert (false / true).

Operator	Bedeutung
<	kleiner
<=	kleiner gleich
>	größer
>=	größer gleich
==	gleich
!=	ungleich

Beispiele:

```
int zahl1, zahl2;
zahl1 = 12;
zahl2 = 30;
zahl1 < zahl2  --> ergibt den logischen Wert true
zahl1 <= zahl2 --> ergibt den logischen Wert true
zahl1 > zahl2  --> ergibt den logischen Wert false
zahl1 >= zahl2 --> ergibt den logischen Wert false
zahl1 == zahl2 --> ergibt den logischen Wert false
zahl1 != zahl2 --> ergibt den logischen Wert true
```

Logische Operatoren

C# bietet eine Reihe logischer Operatoren an, die ebenfalls als Resultat einen booleschen Wert liefern.

Logische Operatoren	
Operator	**Bedeutung**
&&	(**Und-Operator**) Der Ausdruck a && b ist true, wenn sowohl **a** als auch **b** true sind. Zuerst wird **a** ausgewertet. Sollte **a** false sein, ist in jedem Fall der Gesamtausdruck unabhängig von **b** auch falsch. **b** wird dann nicht mehr ausgewertet. .
||	(**Oder-Operator**) Der Ausdruck a || b ist true, wenn entweder **a** oder **b** true ist. Zuerst wird **a** ausgewertet. Sollte **a** bereits true sein, ist in jedem Fall der Gesamtausdruck unabhängig von **b** auch wahr. **b** wird dann nicht mehr ausgewertet.
!	(**Negationsoperator**) Der Ausdruck **!a** ist true, wenn **a** einen unwahren Wert beschreibt, und false, wenn **a** wahr ist.

Das Ergebnis einer Operation, an der logische Operatoren beteiligt sind, lässt sich am besten anhand einer Wahrheitstabelle darstellen.

Wahrheitstabellen				
Bedingung 1 (A)	**Bedingung 2 (B)**	**Und-Operator (A && B)**	**Oder-Operator (A ‖ B)**	**Negationsoperator (!A)**
false	false	false	false	true
true	false	false	true	false
false	true	false	true	
true	true	true	true	

Beispiele:

```
int zahl1, zahl2;
zahl1 = 12;
zahl2 = 30;
(zahl1 < 10) && (zahl2 > 20) --> ergibt den logischen Wert false
(zahl1 < 10) || (zahl2 > 20) --> ergibt den logischen Wert true
(zahl1 <= 20) && (zahl2 > 25) --> ergibt den logischen Wert true
(zahl1 <= 20) || (zahl2 > 25) --> ergibt den logischen Wert true
(zahl1 <= 2) || (zahl2 < 25) --> ergibt den logischen Wert false
```


Verzweigung

Bedingte Anweisung (if) und Verzeigung (if-else)

Oft ist es erforderlich, dass eine Anweisung nur unter einer bestimmten Bedingung ausgeführt wird.

- **if-Anweisung**: Mit der if-Anweisung sorgt man dafür, dass eine Anweisung nur dann ausgeführt wird, wenn eine Bedingung erfüllt ist, also „wahr“ (engl. true) ist.

```
if  (Bedingung)
{
        //diese Anweisungen werden ausgeführt, wenn die Bedingung wahr ist
        …
}
```

- **if-elseAnweisung**: Soll auch etwas passieren, wenn die Bedingung nicht erfüllt ist, d.h. die Bedingung „falsch“ (engl. false) ist, so erweitert man die `if`-Anweisung um eine `else`-Klausel.

```
if  (Bedingung)
{
        //diese Anweisungen werden ausgeführt, wenn die Bedingung wahr ist
        …
}
else
{
        //diese Anweisungen werden ausgeführt, wenn die Bedingung falsch ist
        …
}
```

Beispiel-Programm

Das folgende Programm liest ein Wert von der Eingabe und vergleicht es mit der Null. Wenn der Wert größer als Null ist, so wird auf dem Bildschirm die Zahl mit dem Text „ist größer als 0“ ausgegeben. Falls der Wert nicht größer als Null ist, so wird auf dem Bildschirm die Zahl mit dem Text „ist kleiner gleich 0“ ausgegeben.

```
static void Main(string[] args)
{
    double zahl;

    Console.Write("Wert eingeben: ");
    zahl = Convert.ToDouble(Console.ReadLine());

    if (zahl > 0)
    {
        Console.WriteLine(zahl + " ist größer als 0.");
    }
    else
    {
        Console.WriteLine(zahl + " ist kleiner gleich 0.");
    }
}
```

Nach dem Starten könnte die Bildschirmausgabe so aussehen:

Beachte: Eine Bedingung kann aus mehreren Bedingungen bestehen, diese werden durch UND-Operator (&&) oder ODER-Operatoren (||) verknüpft.

z.B.

- (x > 0) && (y > 0)
- (x > 0) || (y > 0)

Programmieraufgaben (II): if-else-Anweisung

1.Aufgabe **Minimum von zwei Zahlen bestimmen**

- Eingabe: zwei Zahl (**zahl1** und **zahl2**)
- Ausgabe: Minimum der beiden Zahlen, d.h. die kleinste Zahl ausgeben
- z.B.

```
1. Zahl = 7
2. Zahl = 12

Minimum(7; 12) = 12
```

2.Aufgabe **Minimum von drei Zahlen bestimmen**

- Eingabe: drei Zahlen (**zahl1, zahl2** und **zahl3**)
- Ausgabe: Minimum der drei Zahlen
- z.B.

```
1. Zahl = 7
2. Zahl = 12
3. Zahl = 5

Minimum(7; 12; 5) = 5
```

3.Aufgabe **Wurzel ziehen**[4]

- Eingabe: eine Zahl
- Ausgabe: Wurzel der Zahl, falls dieser existiert.
- **Beachte:**
  - Zahl positiv → zwei Wurzellösungen
  - Zahl 0 → eine Lösung 0
  - Zahl negativ → keine Wurzel
- z.B.

```
Zahl = 16

Wurzel(16) = 4 und -4
```

```
Zahl = -5

Wurzel(-5) = existiert nicht
```

[4] Tipp: Wurzel aus einer Zahl **x** zieht man unter C# mit Hilfe der Funktion `Math.Sqrt(x);`

4.Aufgabe **Quadratische Gleichungen ($ax^2 + bx + c = 0$) lösen**

- Eingabe: Koeffizienten **a, b** und **c** der quadratischen Gleichung
- Ausgabe: Lösungen der quadratischen Gleichung

Beachte:

Normalform der quadratischen Gleichung: $x^2 + px + q = 0$

pq-Formel: $x_{1/2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$

Diskriminante: $D = \left(\frac{p}{2}\right)^2 - q$

Die Diskriminante bestimmt die Anzahl der Lösungen einer quadratischen Gleichung.

D > 0	Zwei Lösungen (zwei Nullstellen)
D = 0	Eine Lösung (eine Nullstelle)
D < 0	Keine Lösung (keine Nullstelle)

- z.B.

```
a = 2
b = -8
c = 14
2x^2-8x+14 = 0

x1 = 7
x2 = 1
```

- z.B.

```
a = 1
b = 12
c = 36
1x^2+12x+36 = 0

x1 = -6
```

- z.B.

```
a = 4
b = -24
c = 72
4x^2-24x+72 = 0

Hat keine Lösung!
```


Switch-Anweisung (Mehrfach Verzweigung)

Eine Alternative zur `if-else`-Anweisung stellt die `switch`-Anweisung dar. Sie ermöglicht ebenfalls eine Mehrfachverzweigung, jedoch wirkt der Code wesentlich übersichtlicher und trägt zum besseren Verständnis bei. Die switch-Anweisung prüft eine Variable vom Typ `int` oder `string` auf bestimmte Werte.

Aufbau der switch-Anweisung

```
switch (Variable)
{
    case wert1:
                Anweisung/en;
                break;
    case wert2:
                Anweisung/en;
                break;
:
    default:
                Anweisung/en;
                break;
}
```

Der default-Teil wird ausgeführt, wenn keiner der vorherigen Werte zutrifft. Dieser Teil der switch-Anweisung ist optional, d.h. kann weggelassen werden. Jeder Programmblock muss mit dem Kommando break abgeschlossen werden, damit die switch-Anweisung nicht unnötig weiter durchlaufen wird, wenn es zu einer Übereinstimmung mit einer der case-Werte kommt.

Vergleich zwischen if-else-Befehl und switch-case-Befehl:

if-else

```
int eingabe
..Menü Ausgabe..
-- eingabe lesen...

if (eingabe == 1)
{
      Anweisung/en;
}
else if (eingabe == 2)
{
      Anweisung/en;
}
else if (eingabe == 3)
{
      Anweisung/en;
}
else
{
      Anweisung/en;
}
```

switch-case

```
int eingabe
..Menü Ausgabe..
-- eingabe lesen...

switch(eingabe)
{
      case 1:
            Anweisung/en;
            break;
      case 2:
            Anweisung/en;
            break;
      case 3:
            Anweisung/en;
            break;
      default:
            Anweisung/en;
            break;
}
```

Beispielprogramm[5]

```
   Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
   Console.Write("Please enter your selection: ");

   int n = Convert.ToInt32(Console.ReadLine());
   int cost = 0;

   switch(n)
   {
     case 1:
       cost = 25;
       break;
     case 2:
       cost = 50;
       break;
     case 3:
       cost = 70;
       break;
     default:
       Console.WriteLine("Invalid selection. Please select 1, 2, or
3.");
       break;
   }

   if (cost != 0)
   {
       Console.WriteLine("Please insert {0} cents.", cost);
   }
   Console.WriteLine("Thank you for your business.");
```

Beispielausgabe:

```
Coffee sizes: 1=Small 2=Medium
3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

[5] http://msdn.microsoft.com/de-de/library/06tc147t(v=vs.80).aspx

Schleifen (Wiederholungsanweisungen)

Eine Schleife kommt zum Einsatz, wenn eine Anweisung mehrfach ausgeführt werden soll.

Bedingungsabhängige Schleife (while- und do-while-Schleife)

Bei jedem Schleifendurchgang wird eine Bedingung überprüft. Wenn die Bedingung wahr ist, werden die zur Schleife gehörigen Anweisungen ein weiteres Mal ausgeführt.

Bei der **kopfgesteuerten Schleife (while-Schleife)** wird die Bedingung vor Beginn eines Durchgangs
geprüft, bei der **fußgesteuerten Schleife (do-while-Schleife)** hingegen am Ende.

- **while-Schleife** (kopfgesteuerte Schleife)

```
while (Bedingung)
{
        // diese Anweisungen werden wiederholt ausgeführt,
        // solange die Bedingung wahr ist
        …
}
```

- **do-while-Schleife** (fußgesteuerte Schleife)

```
do
{
        // diese Anweisungen werden mindestens einmal ausgeführt
        // und werden wiederholt,
        // solange die Bedingung wahr ist
        …
}
while (Bedingung);
```

Beispiel für eine do-while-Schleife

Das folgende Programm liest zwei Werte von der Eingabe und vergleicht die beiden Zahlen und gibt den größeren Wert aus.

Der Benutzer kann den Vorgang auf Wunsch wiedeholen.

```
static void Main(string[] args)
{
   double zahl1, zahl2, max;
   int auswahl;

   do
   {
      Console.WriteLine("\n--- Maximum zweier Zahlen berechnen ---");
      Console.Write("1. Zahl eingeben = ");
      zahl1 = Convert.ToDouble(Console.ReadLine());
      Console.Write("2. Zahl eingeben = ");
      zahl2 = Convert.ToDouble(Console.ReadLine());

      if(zahl1 > zahl2)
      {
         max = zahl1;
      }
      else
      {
         max = zahl2;
      }

      Console.WriteLine("\n max( "+ zahl1 +" , " + zahl2 + " ) = " +
max);

      Console.Write("\n\n Wollen Sie wiederholen (ja=1/nein =0)? : ");
      auswahl= Convert.ToInt32(Console.ReadLine());
   }
   while(auswahl == 1);
}
```


Programmieraufgaben (III): while- und do-while-Schleife

1.Aufgabe **Ausgabe von 1 bis n**

- Eingabe: eine ganze Zahl (**n**)
- Ausgabe: alle Zahlen von **1** bis **n** werden ausgegeben.
- z.B.:

```
n eingeben: 5

1 2 3 4 5
```

2.Aufgabe **Ausgabe von n bis 1**

- Eingabe: eine ganze Zahl (**n**)
- Ausgabe: alle Zahlen von **n** bis **1** werden ausgegeben.
- z.B.:

```
n eingeben: 5

5 4 3 2 1
```

3.Aufgabe **Ausgabe von m bis n**

- Eingabe: zwei ganze Zahl (**m** und **n**)
- Ausgabe: alle Zahlen von **m** bis **n** werden ausgegeben.
- z.B.:

```
m eingeben: 3
n eingeben: 9

3 4 5 6 7 8 9
```

4.Aufgabe **Ausgabe m-er Reihe bis n**

- Eingabe: zwei ganze Zahl (**m** und **n**)
- Ausgabe: **m**-er Reihe wird ausgegeben bis **n**.
- z.B.:

```
m eingeben: 4
n eingeben: 21

4 8 12 16 20
```

5.Aufgabe **Sterne ausgeben**

- Eingabe: eine ganze Zahl (**n**)
- Ausgabe: **n** Sterne werden ausgegeben
- z.B.:

```
n eingeben: 4

****
```

6.Aufgabe **Summe von 1 bis n**

- Eingabe: eine ganze Zahl (**n**)
- Ausgabe: alle Zahlen von 1 bis **n** werden addiert und die Summe wird ausgegeben.
- z.B.:

```
n eingeben: 5

Die Summe von 1 bis 5 beträgt 15.
```

7.Aufgabe **Summe von m bis n**

- Eingabe: zwei ganze Zahl (**m** und **n**)
- Ausgabe: alle Zahlen von **m** bis **n** werden addiert und die Summe wird ausgegeben.
- z.B.:

```
m eingeben: 5
n eingeben: 7

Die Summe von 5 bis 7 beträgt 18.
```

8.Aufgabe **Produkt von 1 bis n**

- Eingabe: eine ganze Zahl (**n**)
- Ausgabe: alle Zahlen von **1** bis **n** werden multipliziert und das Produkt wird ausgegeben.
- z.B.:

```
n eingeben: 4

Das Produkt von 1 bis 4 beträgt 24.
```

9.Aufgabe **Fakultät**

- Eingabe: eine ganze Zahl
- Ausgabe: Fakultät dieser Zahl
- z.B.:

```
n eingeben: 4

4! = 24
```

Was ist die Fakultät?

- $0! = 1$ (Sprechweise: „Null Fakultät ist gleich 1“)
- $1! = 1$ (Sprechweise: „1 Fakultät ist gleich 1“)
- $2! = 1 \cdot 2 = 2$
- $3! = 1 \cdot 2 \cdot 3 = 6$
- $4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$
- $5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$
- …
- $n! = 1 \cdot 2 \cdots n$ (für eine positive ganze Zahl n)
- Für negative Zahlen ist die Fakultät nicht definiert.

Zählergesteuerte Schleife (for-Schleife)

Bei zählergesteuerten Schleifen steht die Anzahl der Wiederholungen typischerweise schon vor Schleifenbeginn fest. Zur Ablaufsteuerung wird am Anfang eine Zählvariable initialisiert und dann bei jedem Durchlauf aktualisiert (z.B. inkrementiert). Der zur Schleife gehörige Anweisungsblock wird ausgeführt, solange die Zählvariable eine festgelegte Grenze nicht überschreitet.

```
for (Initialisierung; Bedingung;  Aktualisierung)
{
        // diese Anweisungen werden wiederholt ausgeführt,
        // solange die Bedingung wahr ist
        …
}
```

Zu den drei Bestandteilen der Schleifensteuerung sind einige Erläuterungen erforderlich:

- **Initialisierung:** In der Regel wird man sich auf eine Indexvariable beschränken und dabei einen Ganzzahltyp (`int`) wählen. Diese Variable wird auf einen Startwert gesetzt und falls nötig dabei auch deklariert.
  Deklarationen und Initialisierungen werden vor dem ersten Durchlauf ausgeführt. Eine im Initialisierungsteil deklarierte Variable ist lokal bzgl. der for-Schleife, steht also nur in deren Anweisung(sblock) zur Verfügung.
- **Bedingung:** Üblicherweise wird eine Ober- oder Untergrenze für die Indexvariable gesetzt, doch erlaubt C# beliebige logische Ausdrücke.
  Die Bedingung wird zu Beginn eines Schleifendurchgangs geprüft. Resultiert der Wert true, so findet eine weitere Wiederholung des Anweisungsteils statt, anderen falls wird die for-Anweisung verlassen. Folglich kann es auch passieren, dass überhaupt kein Durchlauf zustande kommt.
- **Aktualisierung:** Die Aktualisierung wird am Ende eines Schleifendurchgangs (nach Ausführung der Anweisung) vorgenommen.

Beispiel für eine for-Schleife

Das folgende Programm liest ein Wert (n) von der Eingabe und gibt die Werte von 1 bis n aus.

```
static void Main(string[] args)
{
   int n, i;

   Console.Write("Zahl eingeben = ");
   n = Convert.ToInt32(Console.ReadLine());

   for (i = 1; i <= n; i++[6])
   {
       Console.Write(i + " ");
   }
}
```

- z.B.:

```
Zahl eingeben: 4

1 2 3 4
```

Folgendes Programm leistet das gleiche wie das obige Programm, wobei hier eine while-Schleife verwendet wurde:

```
static void Main(string[] args)
{
   int n, i;

   Console.Write("Zahl eingeben = ");
   n = Convert.ToInt32(Console.ReadLine());

   i = 1;
   while(i <= n)
   {
        Console.Write(i + " ");
        i++;
   }
}
```

- z.B.:

```
Zahl eingeben: 4

1 2 3 4
```

- [6] **Inkrementierung: `i++;`** ⇔ **`i=i+1;`**
  (Wert wird um eins erhöht)
- **Dekrementierung: `i--;`** ⇔ **`i=i-1;`**
  (Wert wird um eins erniedrigt)

Programmieraufgaben (IV): for-Schleife

1. **Ausgabe von 1 bis n**
   - Eingabe: eine ganze Zahl (**n**)
   - Ausgabe: alle Zahlen von **1** bis **n** werden ausgegeben.
   - z.B.:

```
n eingeben: 5

1 2 3 4 5
```

2. **Ausgabe von n bis 1**
   - Eingabe: eine ganze Zahl (**n**)
   - Ausgabe: alle Zahlen von **n** bis **1** werden ausgegeben.
   - z.B.:

```
n eingeben: 5

5 4 3 2 1
```

3. **Ausgabe von m bis n**
   - Eingabe: zwei ganze Zahl (**m** und **n**)
   - Ausgabe: alle Zahlen von **m** bis **n** werden ausgegeben.
   - z.B.:

```
m eingeben: 3
n eingeben: 9

3 4 5 6 7 8 9
```

4. **Ausgabe m-er Reihe bis n**
   - Eingabe: zwei ganze Zahl (**m** und **n**)
   - Ausgabe: **m**-er Reihe wird ausgegeben bis **n**.
   - z.B.:

```
m eingeben: 4
n eingeben: 21

4 8 12 16 20
```

5. **Summe von m bis n**
   - Eingabe: zwei ganze Zahl (**m** und **n**)
   - Ausgabe: alle Zahlen von **m** bis **n** werden addiert und die Summe wird ausgegeben.
   - z.B.:

```
m eingeben: 5
n eingeben: 7

Die Summe von 5 bis 7 beträgt 18.
```

Programmieraufgaben (V): Zahlenratespiel

Schreiben Sie ein Programm in C#, mit deren Hilfe ein Benutzer eine ganze Zahl zwischen 1 und 1000 erraten kann.

Beispiel:

```
---- Zahlenraten zwischen 1 und 1000
1. Versuch: 500
                --> zu klein
2. Versuch: 750
                --> zu gross
3. Versuch: 600
                --> zu gross
4. Versuch: 550
                --> zu gross
5. Versuch: 525
                --> zu gross
6. Versuch: 510
                --> zu klein
7. Versuch: 515
                --> zu klein
8. Versuch: 520
                --> zu klein
9. Versuch: 523
                --> erraten
Sie haben  9 Versuche gebraucht!
```

Programmrahmen: Zahlenratespiel

```
class Program
{
        static void Main(string[] args)
        {
            // - Erzeuge eine Zufallszahl zwischen 1 und 1000-

            Random zzahl = new Random();
            int zufallszahl = zzahl.Next(1, 1000);

            // ------------------------------------


             // Schreiben Sie hier Ihren Programm



        }
}
```


Programmieraufgaben (VI):Wurzel ziehen mit Intervallhalbierungsverfahren[7]

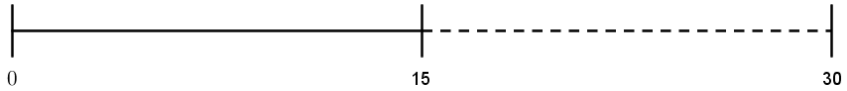
Wie berechnet man die Wurzel einer Zahl a > 0?

Wie berechnet der Taschenrechner den Wert $\sqrt{30} \approx 5{,}477225575$?

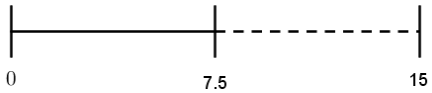
Wurzel ziehen mit dem Intervallhalbierungsverfahre

Die „Wurzel aus 30“ ($\sqrt{30}$) liegt zwischen 0 und 30.

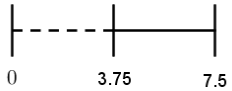
1. Schritt:
   i. Mitte aus 0 und 30 bestimmen → $\frac{0+30}{2} = 15$
   ii. $15^2 = 225 > 30$ => $\sqrt{30}$ liegt zwischen **0** und **15**.



2. Schritt:
   i. Mitte aus 0 und 15 bestimmen → $\frac{0+15}{2} = 7{,}5$
   ii. $7{,}5^2 = 56{,}25 > 30$ => $\sqrt{30}$ liegt zwischen **0** und **7,5**.



3. Schritt:
   i. Mitte aus 0 und 7,5 bestimmen → $\frac{0+7{,}5}{2} = 3{,}75$
   ii. $3{,}75^2 = 14{,}0625 < 30$ => $\sqrt{30}$ liegt zwischen **3,75** und **7,5**.

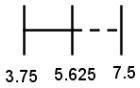


[7] Die **Bisektion**, auch **Intervallhalbierungsverfahren** genannt, ist ein Verfahren der Mathematik und der Informatik. Durch sie wird eine konvergente Folge von Intervallschachtelungen erzeugt. Das Wort setzt sich zusammen aus *Bi* „Zwei“ und *Sektion* „Schnitt“. Es steht also für Zwei-Teilung. (http://de.wikipedia.org/wiki/Bisektion)

4. Schritt:

   i. Mitte aus 3,75 und 7,5 bestimmen → $\frac{3{,}75+7{,}5}{2}=5{,}625$

   ii. $5{,}625^2 = 31{,}640625 > 30$ => $\sqrt{30}$ liegt zwischen **3,75** und **5,625**.



- Usw..

Beachte:

- 1.Fall : $a \geq 1$: Wurzel von a liegt zwischen 0 und a $(0 \leq \sqrt{a} \leq a)$
- 2. Fall : $0 < a < 1$: Wurzel von a liegt zwischen 0 und 1 $(0 < \sqrt{a} < 1)$
  Die Wurzel von a liegt sogar zwischen a und 1 $(a < \sqrt{a} < 1)$.

Aufgabe: Schreiben Sie ein Programm in C#, mit deren Hilfe man die Wurzel einer Zahl ziehen kann. Verwenden Sie das Halbierungsverfahren.

Der Benutzer gibt zwei Zahlen ein. Von der ersten Zahl soll die Wurzel gezogen werden. Die zweite Zahl gibt an, wie oft das Halbierungsverfahren angewandt werden soll. (Anzahl der Iterationen)

z.B.:

```
----------------------------------------
 Wurzel ziehen mit Halbierungsverfahren
----------------------------------------

Zahl eingeben = 30
Anzahl der Iterationen eingeben = 10

----------------------------------------

1.Schritt: 0 <= Wurzel(30) <= 15
2.Schritt: 0 <= Wurzel(30) <= 7,5
3.Schritt: 3,75 <= Wurzel(30) <= 7,5
4.Schritt: 3,75 <= Wurzel(30) <= 5,625
5.Schritt: 4,6875 <= Wurzel(30) <= 5,625
6.Schritt: 5,15625 <= Wurzel(30) <= 5,625
7.Schritt: 5,390625 <= Wurzel(30) <= 5,625
8.Schritt: 5,390625 <= Wurzel(30) <= 5,5078125
9.Schritt: 5,44921875 <= Wurzel(30) <= 5,5078125
10.Schritt: 5,44921875 <= Wurzel(30) <= 5,478515625


 ---> Wurzel(30) = 5,4638671875
----------------------------------------
```


Programmieraufgaben (VII): Weitere Aufgaben

1. Aufgabe: Schreiben Sie ein C#-Programm, das zwei positive ganze Zahlen einliest. Die Eingabe der beiden Zahlen soll solange wiederholt werden bis der Benutzer zwei positive Zahlen eingibt.
Der Abstand zwischen den beiden Zahlen soll berechnet und ausgegeben werden.

Die Ein- und Ausgabe soll wie folgt aussehen:

1. Beispiel:

```
1. Zahl = 3
2. Zahl = -1
1. Zahl = -2
2. Zahl = 3
1. Zahl = -3
2. Zahl = -4
1. Zahl = 2
2. Zahl = 8
Abstand = 6
```

2. Beispiel:

```
1. Zahl = 6
2. Zahl = 1
Abstand = 5
```

Beachte: Der Abstand zweier Zahlen ist immer positiv.

2. Aufgabe: **Umwandlung: Kartesische Form ⇔ Eulersche Form**

Schreiben Sie ein Programm in C#, das eine komplexe Zahlen in der kartesischen Form einliest und in die Exponentialform (Eulersche Form) umwandelt.

z.B.:

```
Umwandlung eine komplexen Zahl von der kartesischen in die Exponentialform

Realteil eingeben (a): 3
Imaginärteil eingeben (b): 5

 Kartesische Form: 3 + 5 i

 Exponentialform (in DEG): 5,8309518948453 * e^(59,04° i)

 Exponentialform (in RAD): 5,8309518948453 * e^(1,03 i)
```

Beachte:

- Wurzel aus einer Zahl **x** zieht man unter C# mit Hilfe der Funktion `Math.Sqrt(x);`
- Die Kreiszahl Pi erhält man durch `Math.PI;`
- Arkustangens von x“ $\arctan(x) = \tan^{-1}(x)$ erhält man durch `Math.Atan(x);` wobei der Winkel in Bogenmaß zurückgegeben wird.

3. Aufgabe: Schreiben Sie ein Programm in C#, das eine komplexe Zahlen in der eulerschen Form einliest und in die kartesische Form umwandelt.

Programmieraufgaben (VIII): Einmaleins / Grundrechenarten

1.Aufgabe: **(Das kleine Einmaleins)**

Schreiben Sie ein Programm, um das "Kleine Einmaleins" zu üben.
Das Programm bestimmt zwei zufällige Zahlen[8] zwischen 1 und 10.
Dann wird aus diesen beiden Zahlen eine Aufgabe auf der Konsole ausgegeben. Der Benutzer gibt die Lösung ein und wenn die Eingabe des Benutzers richtig ist, so wird "Richtig!" ausgegeben. Falls die Eingabe falsch ist, so wird "Falsch!" ausgegeben.

Beispiel:

```
--- Kleine Einmaleins üben ---

3*5 = 15
Richtig!  (1x richtig; 0xfalsch)

2*10 = 20
Richtig!  (2x richtig; 0xfalsch)

4*8 = 16
Falsch!  (2x richtig; 1xfalsch)
:
```

Ihr Programm soll immer wieder neue Aufgaben stellen. Das Programm soll abbrechen, wenn..

a) ..der Benutzer 3-mal eine richtige oder 3-mal eine falsche Lösung eingegeben hat. D.h. nach maximal 5 Aufgaben wird das Programm beendet.

b) ..der Benutzer 3-mal hintereinander eine richtige oder 3-mal hintereinander eine falsche Lösung eingegeben hat.

[8]
```
// Erzeuge eine Zufallszahl zwischen 1 und 10
Random zzahl = new Random();
int zufallszahl = zzahl.Next(1, 10);
```

c) Erweitern Sie ihr Programm, so dass die Aufgabe gezählt werden und am Ende eine Statistik ausgegeben wird, wie viel Prozent der Aufgabe richtig bzw. falsch gelöst worden sind.

Beispiel:

```
--- Kleine Einmaleins üben ---

1.Aufgabe:  3*5 = 15
Richtig!  (1x richtig; 0xfalsch)

2.Aufgabe:  2*10 = 20
Richtig!  (2x richtig; 0xfalsch)

3.Aufgabe:  4*8 = 16
Falsch!  (2x richtig; 1xfalsch)

4.Aufgabe:  3*9 = 27
Richtig! (3x richtig; 1xfalsch)

5.Aufgabe:  8*2 = 16
Richtig! (4x richtig; 1xfalsch)

6.Aufgabe:  7*3 = 21
Richtig! (5x richtig; 1xfalsch)

Richtig = 5/6 = 83,33%
Falsch = 1/6 = 16,67%

Programm wird beendet!
```


2.Aufgabe: **(Grundrechenarten)**

Schreiben Sie ein Programm, um die Grundrechenarten zu üben.
Das Programm gibt ein Menü aus. Der Benutzer kann eine Rechenart (Addition, Subtraktion, Multiplikation, Division) auswählen.
Anschließend gibt der Benutzer zwei Zahlen ein und die Rechnung wird ausgegeben.

Beispiel:

```
Grundrechenarten
-----Menü-----
   (1)     Addition
   (2)     Subtraktion
   (3)     Multiplikation
   (4)     Division
---------------
Ihre Wahl:  3
---------------

1.Zahl eingeben = 30
2.Zahl eingeben = 4

=>  30*4 = 120
```

3.Aufgabe: **(Grundrechenarten mit Wiederholung)**
Erweitern Sie ihr Programm von der 2.Aufgabe:
Die Berechnung soll nun wiederholt werden bis der Benutzer im Menü die 0 wählt. Falls der Benutzer eine falsche Wahl trifft, so soll die Eingabe wiederholt werden.
Beispiel:

```
Grundrechenarten

-----Menü-----
   (1)     Addition
   (2)     Subtraktion
   (3)     Multiplikation
   (4)     Division
   (0)     ENDE
---------------
Ihre Wahl:  5
---------------

Falsche Eingabe!


-----Menü-----
   (1)     Addition
   (2)     Subtraktion
   (3)     Multiplikation
   (5)     Division
   (0)     ENDE
---------------
Ihre Wahl: 1
---------------

1.Zahl eingeben = 30
2.Zahl eingeben = 4

=>  30+4 = 34


-----Menü-----
   (1)     Addition
   (2)     Subtraktion
   (3)     Multiplikation
   (4)     Division
   (0)     ENDE
---------------
Ihre Wahl: 0
---------------

Programm wurde beendet!
```


Programmieraufgaben (IX): von A.Fey und Z.Genc

1. Überprüfen der Bestellung

Bobs Schrauben Discount verlangt folgende Preise:

a) pro Schraube 5 Cent
b) pro Mutter 3 Cent
c) pro Unterlegscheibe 1 Cent

Schreiben Sie ein Programm, das den Anwender nach der Anzahl der Schrauben, Muttern und Unterlegscheiben fragt und dann den Gesamtbetrag berechnet und ausgibt. Zusätzlich überprüft das Programm die Bestellung. Es ist normalerweise ein Fehler, wenn mehr Schrauben als Muttern bestellt werden. In diesem Fall gibt das Programm die Meldung aus „**Kontrollieren Sie Ihre Bestellung!**“ Andernfalls gibt das Programm aus „**Die Bestellung ist okay**“. In jedem Fall wird der Gesamtbetrag ausgegeben.

```
Anzahl der Schrauben: 12
Anzahl der Muttern: 8
Anzahl der Unterlegscheiben: 24

Kontrollieren Sie Ihre Bestellung!
Gesamtbetrag: 1,08 Euro
```

2. Jahr 2000-Problem

Schreiben Sie ein Programm, dass den Anwender nach seinem Geburtsjahr fragt, kodiert als zwei Ziffern (z.B.: 62), und nach dem aktuellen Jahr fragt, ebenfalls kodiert als zwei Ziffern (99). Das Programm soll das richtige Alter des Anwenders in Jahren ausgeben.

```
Geburtsjahr: 62
Aktuelles Jahr: 99
Ihr Alter ist: 37
```

```
Geburtsjahr: 62
Aktuelles Jahr: 04
Ihr Alter ist: 42
```

3. Wettbewerb Torten essen

Beim *State Fair Pie Eating Contest* müssen alle Teilnehmer in der Schwergewichtsklasse zwischen 235 und 265 Pfund wiegen. Schreiben Sie ein Programm, das nach dem Gewicht des Teilnehmers fragt und dann ausgibt, ob er zum Wettbewerb zugelassen ist oder nicht.

4. Letzte Tankmöglichkeit vor Death Valley

Als *Last Chance Gas Station* befindet sich an der Straße 190 am Rande des *Death Valley*. Die nächsten 200 km gibt es keine weitere Tankstelle. Sie sollen ein Programm schreiben, das den Fahrern hilft zu entscheiden, ob sie tanken sollen oder nicht. Das Programm fragt nach…

a) … der Tankkapazität in Litern,

b) …der Benzinanzeige in Prozent (voll= 100, drei viertel voll = 75 usw.), sowie

c) …dem Benzinverbrauch des Fahrzeuges in Kilometern pro Liter.

Die Ausgabe des Programms ist „**Tanken**!“ oder „**Weiterfahren**“. Je nachdem, ob das Fahrzeug genug Benzin für 200 Kilometer hat oder nicht.

```
Tankkapazitaet: 12
Benzinanzeige: 50
km pro Liter: 30

Tanken!
```

5. Delikatessen

Sam & Ellas Delikatessen brauchen ein Programm, das Bestellungen über ein Internetformular entgegen nimmt. Das Programm fragt welchen Artikel der Kunde möchte, den Preis und ob der Artikel per Express verschickt werden soll. Für Artikel unter 10 € betragen die Versandkosten 3 €. Kostet der Artikel 10 € oder mehr betragen sie 2 €. Der Expresszuschlag beträgt 5 €.

```
Geben Sie den Artikel ein: Schoko
Geben Sie den Preis in Euro ein: 12
Expressversand (0==nein, 1==ja): 1

Rechnung
Schoko: 12.00
Versand: 7.00
Gesamt: 19.00
```

6. Scheckgebühr

Eine Bank verfährt nach der folgenden Regel: Wenn ein Kunde auf seinem Girokonto ein Guthaben von mehr als 1000 € oder auf seinem Sparkonto ein Guthaben von mehr als 1500 € hat, wird keine Scheckgebühr erhoben. Andernfalls wird eine Gebühr von 0.50 € erhoben. Schreiben Sie ein Programm, das nach dem Kontostand der beiden Konten fragt und dann ausgibt, ob eine Gebühr erhoben wird oder nicht.

7. Fantasy-Spiel

In einem Fantasy-Rollenspiel können die Spieler ihre Spielfiguren entwerfen, indem Sie einen Punktwert für jedes von drei Merkmalen vergeben:

a) Stärke, von 1 bis 10

b) Gesundheit, von 1 bis 10

c) Glück, von 1 bis 10

Schreiben Sie ein Programm, das nach dem Namen der Spielfigur und nach den Punktwerten der einzelnen drei Merkmale fragt. Aber, die Gesamtpunktzahl muss geringer als 15 sein. Wenn die Gesamtpunktzahl 15 überschreitet, dann werden jedem Merkmal 5 Punkte zugewiesen.

```
Willkommen zu Yertle's Schatzsuche
Geben Sie den Namen der Spielfigur ein:
Chortle
Staerke eingeben (1-10): 8
Gesundheit eingeben (1-10): 4
Glueck eingeben (1-10): 6

Sie haben Ihrer Spielfigur zu viele Punkte gegeben!
Es wurden die Standardwerte zugewiesen:
Chortle, Staerke: 5, Gesundheit: 5, Glueck: 5
```

8. Reifendruck 1

Die beiden vorderen Reifen eines Autos sollten beide den gleichen Reifendruck haben. Ebenso sollten die beiden hinteren Reifen den gleichen Reifendruck haben, aber nicht unbedingt den gleichen wie die Vorderreifen. Schreiben Sie ein Programm, das den Reifendruck der vier Reifen einliest und dann eine Meldung ausgibt, ob der Reifendruck in Ordnung ist.

```
Reifendruck: rechter Vorderreifen : 2.2
Reifendruck: linker Vorderreifen : 2.2
Reifendruck: rechter Hinterreifen : 2.0
Reifendruck: linker Hinterreifen : 2.0

Reifendruck ist OK
```

9. Reifendruck 2

Es genügt nicht, dass der Reifendruck der beiden Vorder- und Hinterräder der gleiche ist. Beide Reifendrücke müssen sich innerhalb eines bestimmten Bereichs befinden. Erweitern Sie das Programm aus der letzten Aufgabe. Es soll überprüfen, ob sich der Druck aller Reifen zwischen 1.9 und 2.2 bar befindet. Befindet sich ein Reifen außerhalb dieses Bereichs, wird am Schluss eine Warnmeldung ausgegeben.

```
Reifendruck: rechter Vorderreifen : 1.7
Reifendruck: linker Vorderreifen : 1.7
Reifendruck: rechter Hinterreifen : 2.2
Reifendruck: linker Hinterreifen : 2.2

Reifendruck rechts vorne, links vorne ist nicht OK
```

Deklarieren Sie dazu für jeden Reifen eine boolesche Variable **druckOK**. Initialisieren Sie diese mit **true** und setzen Sie den Wert auf **false**, wenn sich ein Reifen außerhalb des gültigen Bereichs befindet.

10. Reifendruck 3

Reifen müssen nicht genau den gleichen Druck haben. Verändern Sie Ihr Programm aus der letzten Aufgabe, so dass sich die beiden Vorder- und Hinterräder in einem Toleranzbereich von 0,2 bar bewegen dürfen.

```
Reifendruck: rechter Vorderreifen : 2.1
Reifendruck: linker Vorderreifen : 2.0
Reifendruck: rechter Hinterreifen : 2.2
Reifendruck: linker Hinterreifen : 2.2

Reifendruck ist OK
```

11. Zahlenausgabe

Schreiben Sie ein Programm, das nach Eingabe eines Start- und eines Endwertes alle Zahlen (Integer) inklusive der eingegebenen ausgibt!

```
Startwert: 5
Endwert: 9

5  6  7  8  9
```


12. Zahlen addieren

Schreiben Sie ein Programm, das die Zahlen addiert, die vom Benutzer eingegeben werden. Das Programm fragt zuerst wie viele Zahlen addiert werden sollen. Danach fordert das Programm den Benutzer auf die Zahlen nacheinander einzugeben. Schließlich gibt es das Ergebnis auf den Bildschirm aus. Die Zahlen sollen vom Typ Integer sein.

```
Wie viele Zahlen sollen addiert werden? 5
Geben Sie eine Zahl ein: 3
Geben Sie eine Zahl ein: 4
Geben Sie eine Zahl ein: -4
Geben Sie eine Zahl ein: -3
Geben Sie eine Zahl ein: 7

Die Summe ist 7
```

Geben Sie auch Dezimalzahlen ein und erklären Sie ggf. die Unterschiede!

13. Summe 1/n

Schreiben Sie ein Programm, das die folgende Summe berechnet:

$$summe = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$$

Die Variable **n** ist eine Integer-Zahl, bis zu dem die Reihe berechnet werden soll. Sie wird vom Anwender eingegeben.[9]

```
Geben Sie n ein: 4

Die Summe ist: 2.083333333
```

14. Addition von Quadrat- und Kubikzahlen

Schreiben Sie ein Programm, das Quadrat- und Kubikzahlen (Integer) von 1 bis n addiert. Wobei n vom Anwender eingegeben wird.

```
Geben Sie n ein: 5
Die Quadratsumme ist: 55
Die Kubiksumme ist: 225
```

Verwenden Sie nur eine Schleife.

[9] **Beachte:** 1/k liefert für eine ganze Zahl (int) k größer als 1 immer die Null. 1.0/k liefert ein richtiges Ergebnis oder man definiert die Variable k als double (Dezimalzahl) oder man konvertiert die ganze Zahlen in Dezimalzahlen.

15. Potenz einer Zahl[10]

Schreiben Sie ein Programm, das x^n berechnet, wobei x eine Dezimalzahl und n eine positive ganze Zahl ist. Bei der Eingabe einer negativen Zahl, soll eine Fehlermeldung erfolgen.

$$x^n = x \cdot x \cdot x \cdot \ldots \cdot x \qquad \text{(n-mal)}$$

Der Anwenderdialog soll ungefähr so aussehen:

```
Geben Sie x ein: 1.3
Geben Sie n ein: 5

1.3 hoch 5 = 3.7129300000000014
```

```
Geben Sie x ein: 5.6
Geben Sie n ein: -3

n muss ein positive ganze Zahl sein.
```

16. Potenz einer Zahl2

Erweitern Sie das Programm aus der letzten Aufgabe so, dass auch negative Exponenten eingegeben werden dürfen.

```
Geben Sie x ein: 2.5
Geben Sie n ein: -4

2.5 hoch -4 = 0.0256
```

17. Keil von Sternen

Schreiben Sie ein Programm, das Sterne in Keilform ausgibt. Der Anwender gibt die Anfangszahl für die Sterne ein und das Programm gibt Zeilen von Sternen aus, wobei jede Zeile einen Stern weniger hat als die vorhergehende.

```
Anfangszahl der Sterne: 7

*******
******
*****
****
***
**
*
```

[10] Lösen Sie die Aufgabe mit einer Schleife und verwenden Sie nicht vorhandene mathematische Funktionen.

18. Weihnachtsbaum

Schreiben Sie ein Programm, das einen Baum aus Sternen auf den Bildschirm ausgibt.

a) Die Eingabe sollte sinnvolle Werte annehmen.

b) Der Benutzer bestimmt die Höhe des Baumes.

c) Der Stamm sollte etwa halb so groß sein, wie die Höhe des Baumes.

d) Die Breite des Stammes sollte immer drei Sterne betragen und sich in der Mitte des Baumes befinden.

```
Hoehe des Baumes: 9

    *
   ***
  *****
 *******
*********
   ***
   ***
   ***
   ***
```

19. Kilometer pro Liter

Schreiben Sie ein Programm, das die gefahrenen Kilometer pro Liter für eine Reihe von Autos berechnet. Die Daten für jedes Auto bestehen aus dem Anfangs- und Endstand in Kilometern und der Anzahl der Liter. Das Programm wird beendet, sobald der Anwender eine negative Zahl als Anfangsstand eingibt.

```
Programm km pro Liter
Anfangsstand km: 15000
Endstand km: 15250
Liter: 10
km pro Liter: 25.0

Anfangsstand km: 107000
Endstand km: 107450
Liter 15
km pro Liter: 30.0

Anfangsstand km: -1
```

20. Bereiche addieren

Schreiben Sie ein Programm, das den Anwender nach einer Unter- und Obergrenze für einen Bereich von ganzen Zahlen fragt. Das Programm fragt dann den Anwender nach ganzen Zahlen, die addiert werden sollen. Das Programm berechnet zwei Summen:

a) Die Summe der Integer, die im Bereich sind (inklusive)

b) und die Summe der Integer, die außerhalb des Bereichs sind.

Der Anwender signalisiert das Ende der Eingabe mit einer 0.

```
Bereiche addieren
Untergrenze des Bereichs: 20
Obergrenze des Bereichs: 50
Daten eingeben: 21
Daten eingeben: 60
Daten eingeben: 49
Daten eingeben: 30
Daten eingeben: 91
Daten eingeben: 0

Summe der Werte innerhalb des Bereichs: 100
Summe der Werte ausserhalb des Bereichs: 151
```

21. Versandkosten-Kalkulator

Ein Paketdienst berechnet 3.00 Euro Versandkosten bis zu einem Gewicht von 10 Pfund (inklusive). Darüber sind für jedes Pfund zusätzlich 0.25 Euro zu bezahlen. Schreiben Sie ein Programm, das den Anwender nach dem Gewicht der Sendung fragt und dann die Versandkosten ausgibt. Das Programm endet, wenn ein Gewicht von 0 oder weniger eingegeben wird.

```
Gewicht des Sendung: 5
Versandkosten: 3.0 Euro

Gewicht des Sendung: 20
Versandkosten: 5.5 Euro

Gewicht des Sendung: 0
```

22. Abbruchbedingungen

Erklären Sie, was bei den folgenden Schleifen passiert!

a) **while(i<0) k=2*k;**

b) **while(i!=0) i=i-2;**

c) **while(n!=1) {i++; n=2*i;}**

23. Fläche von Rechtecken

Ein CAD Programm erwartet, dass der Anwender die Koordinaten von zwei Eckpunkten für jedes von mehreren Rechtecken eingibt (siehe Diagramm). Dabei wird vorausgesetzt, das die Seiten der Rechtecke parallel zur X und Y Achse verlaufen. Die Koordinaten eines jeden Eckpunkts werden als Paar von Ganzzahlen eingegeben, zuerst die X-Koordinate und dann die Y-Koordinate. Der Ursprung des Koordinatensystems (0,0) ist die linke obere Ecke, so dass Y abwärts zunimmt und X nach rechts zunimmt.

Das Programm berechnet für jedes Rechteck die Höhe, die Breite und die Fläche. Die zwei eingegebenen

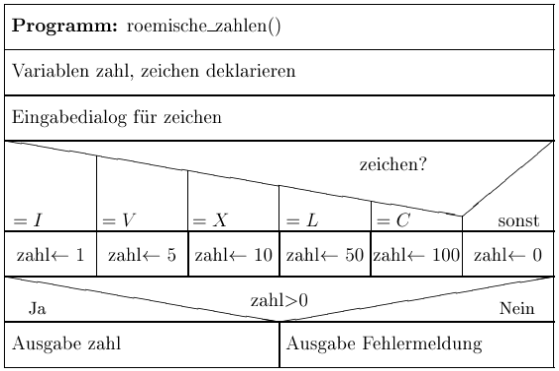
Eckpunkte müssen sich diagonal gegenüber liegen (links oben und rechts unten, oder rechts oben und links unten), aber welche Wahl für jedes Rechteck getroffen wird, ist Sache des Anwenders. Der Anwender kann die Eckpunkte in beliebiger Reihenfolge eingeben. Höhe und Breite sind immer positive (das Programm wird seine Berechnungen anpassen müssen, damit das wahr ist.)

Das Programm wird sanft beendet, wenn der Anwender Eckpunkte eingibt, die nicht die eines Rechtecks sein können (entweder die Höhe ist 0, die Breite ist 0 oder beide.)

```
CAD Programm
Erster Eckpunkt X-Koordinate: 100
Erster Eckpunkt Y-Koordinate: 100
Zweiter Eckpunkt X-Koordinate: 250
Zweiter Eckpunkt Y-Koordinate: 200
Breite:  150  Hoehe: 100   Flaeche: 15000

Erster Eckpunkt X-Koordinate: 250
Erster Eckpunkt Y-Koordinate: 200
Zweiter Eckpunkt X-Koordinate: 100
Zweiter Eckpunkt Y-Koordinate: 100
Breite:  150  Hoehe: 100   Flaeche: 15000

Erster Eckpunkt X-Koordinate: 100
Erster Eckpunkt Y-Koordinate: 200
Zweiter Eckpunkt X-Koordinate: 250
Zweiter Eckpunkt Y-Koordinate: 100
Breite:  150  Hoehe: 100   Flaeche: 15000

Erster Eckpunkt X-Koordinate: 100
Erster Eckpunkt Y-Koordinate: 100
Zweiter Eckpunkt X-Koordinate: 100
Zweiter Eckpunkt Y-Koordinate: 100
Breite:  0  Hoehe: 0   Flaeche: 0

Programm wurde beendet!
```

24. Römische Zahlen

Schreiben Sie ein Programm, das bei Eingabe einer römischen Ziffer den entsprechenden Zahlenwert ausgibt! Das zugehörige Struktogramm ist unten dargestellt.



25. Treppenfunktion

Eine Treppenfunktion besitzt folgende Funktionsvorschrift:

$$f(x) = \begin{cases} 1, & 0 \leq x < 2 \\ 2, & 2 \leq x < 4 \\ 0, & sonst \end{cases}$$

Der Pseudocode dieser Funktion lautet:

```
Falls x größer gleich 0 und kleiner 2 ist
   Dann gilt f(x)=1
Sonst
   Falls x größer gleich 2 und kleiner 4 ist
      Dann gilt f(x)=2
   Sonst gilt f(x)=0
```

Erstellen Sie ein Struktogramm und programmieren Sie diese Funktion! Bei der Eingabe einer Zahl x soll der Funktionswert f(x) ausgegeben werden.

26. Minimum

Programmieren Sie eine Funktion, die zwei Zahlen einliest, diese beiden Zahlen miteinander vergleicht und die kleinere Zahl wieder auf dem Bildschirm ausgibt!

Erstellen Sie das Struktogramm und das Programm!

27. Summe

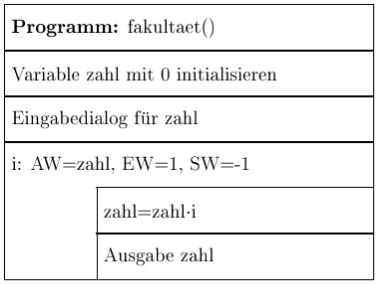
Fertigen Sie ein Programm an, dass zwei Zahlen m, n einliest und die Summe der Zahlen von m bis n berechnet! Falls die Obergrenze kleiner als die Untergrenze ist, sollen die Zahlen ebenfalls addiert werden. Auch hier ist zuerst das Struktogramm zu erstellen!

Die Bildschirmausgabe soll wie folgt aussehen:

```
Untere Grenze: 4
Obere Grenze: 8
4+5+6+7+8=30

Untere Grenze: 3
Obere Grenze: -2
-2+-1+0+1+2+3=3
```

28. Fakultät

Fertigen Sie ein Programm an, dass eine Zahl n einliest und das Produkt der Zahlen von 1 bis n berechnet! Setzen Sie dabei das unten stehende Struktogramm um!



AW=Anfangswert, EW=Endwert, SW=Schrittweite

29. Produkt durch Addition

Das Produkt zweier ganzer Zahlen ist mit Hilfe der Addition zu berechnen. Beachten Sie, dass die Zahlen auch 0 oder negativ sein können.

```
Berechnung des Produkts durch Addition
Multiplikator: 3
Multiplikand: 5
Das Produkt ist 15
```

30. Teiler einer Zahl[11]

Lassen Sie alle Teiler einer einzulesenden ganzen Zahl ausgeben. Benutzen Sie auch hier eine Division mit ganzzahligem Ergebnis.

```
Teiler einer Zahl
Geben Sie eine ganze Zahl ein: 63

63 ist durch 1 3 7 9 21 63 teilbar.
```

31. Rest durch Subtraktion

Der ganzzahlige Quotient $\frac{a}{b}$ zweier ganzer Zahlen a und b und der entstehende Rest sollen über die Subtraktion berechnet werden. Überlegen Sie, welche Fehlerquellen es geben kann und berücksichtigen Sie diese im Programm.

```
Berechnung des Quotienten und Rest durch Subtraktion
Dividend: 25
Divisor: 3
Ganzzahliger Quotient: 8
Ganzzahliger Rest: 1
```

```
Berechnung des Quotienten und Rest durch Subtraktion
Dividend: -25
Divisor: -3
Ganzzahliger Quotient: 8
Ganzzahliger Rest: -1
```

```
Berechnung des Quotienten und Rest durch Subtraktion
Dividend: -25
Divisor: 3
Ganzzahliger Quotient: -8
Ganzzahliger Rest: -1
```

```
Berechnung des Quotienten und Rest durch Subtraktion
Dividend: 25
Divisor: -3
Ganzzahliger Quotient: -8
Ganzzahliger Rest: 1
```

[11] Tipp: Rest einer Division erhält man mit dem Modulo-Operator "%": x = 13%5 => x=3

32. Durchschnitt

Lassen Sie eine beliebige Anzahl von Zahlen addieren, bis die Zahl 0 eingegeben wird. Danach soll die Gesamtsumme und der Durchschnitt ausgegeben werden.

```
Summe und Durchschnitt
Zahl (Ende mit 0): 1.5
Zahl (Ende mit 0): 3
Zahl (Ende mit 0): 1.5
Zahl (Ende mit 0): 0
Summe: 6.0
Durchschnitt: 2.0
```

33. Ungerade Zahlen

Schreiben Sie ein Programm, das die ungeraden Zahlen zwischen zwei einzugebenden Werten ausgibt!

```
Berechnung der ungeraden Zahlen
Untere Grenze: 5
Obere Grenze: 12

Ungerade Zahlen, die zwischen 5 und 12 liegen:
7  9  11
```

34. Quadratzahlen

Es sollen die Quadratzahlen, die Kubikzahlen und die Kehrwerte der Zahlen zwischen zwei einzugebenden Grenzen ausgegeben werden.

```
Untere Grenze: 4
Obere Grenze: 5

Quadratzahl | Kubikzahl | Kehrwert
16.0        | 64.0      | 0.25
25.0        |125.0      | 0.2
```

35. Wirkstoffgehalt eines Arzneimittels

Ein bestimmtes Arzneimittel verliert jeden Monat 4% seines Wirkstoffgehalts. Wenn sein Wirkstoffgehalt unter 50% fällt, muss es ausgemustert werden. Schreiben Sie ein Programm, das berechnet wie viele Monate eine Arznei gelagert werden kann.

```
Monat: 0  Wirkstoffgehalt: 100.0
Monat: 1  Wirkstoffgehalt: 96.0
Monat: 2  Wirkstoffgehalt: 92.16
Monat: 3  Wirkstoffgehalt: 88.47359999999999
Monat: 4  Wirkstoffgehalt: 84.93465599999999
Monat: 5  Wirkstoffgehalt: 81.53726975999999
Monat: 6  Wirkstoffgehalt: 78.27577896959998
Monat: 7  Wirkstoffgehalt: 75.14474781081599
Monat: 8  Wirkstoffgehalt: 72.13895789838334
Monat: 9  Wirkstoffgehalt: 69.253399582448
```

```
Monat: 10  Wirkstoffgehalt: 66.48326359915008
Monat: 11  Wirkstoffgehalt: 63.82393305518407
Monat: 12  Wirkstoffgehalt: 61.27097573297671
Monat: 13  Wirkstoffgehalt: 58.82013670365764
Monat: 14  Wirkstoffgehalt: 56.46733123551133
Monat: 15  Wirkstoffgehalt: 54.20863798609088
Monat: 16  Wirkstoffgehalt: 52.04029246664724
Monat: 17  Wirkstoffgehalt: 49.95868076798135 ABGELAUFEN
```

36. Ratenzahlung

Angenommen Sie schulden Ihrer Kreditkartengesellschaft 1000 €. Die Gesellschaft berechnet Ihnen monatlich 1.5% Zinsen für den gewährten Kredit. Sie entscheiden sich die Kreditkarte nicht mehr zu verwenden und Ihren Kredit monatlich mit einem bestimmten Betrag n zurückzuzahlen. Schreiben Sie ein Programm, das nach dem monatlichen Rückzahlungsbetrag fragt, dann den Saldo und den Gesamtbetrag der Zahlungen für jeden Monat ausgibt, bis der Saldo Null oder weniger beträgt.

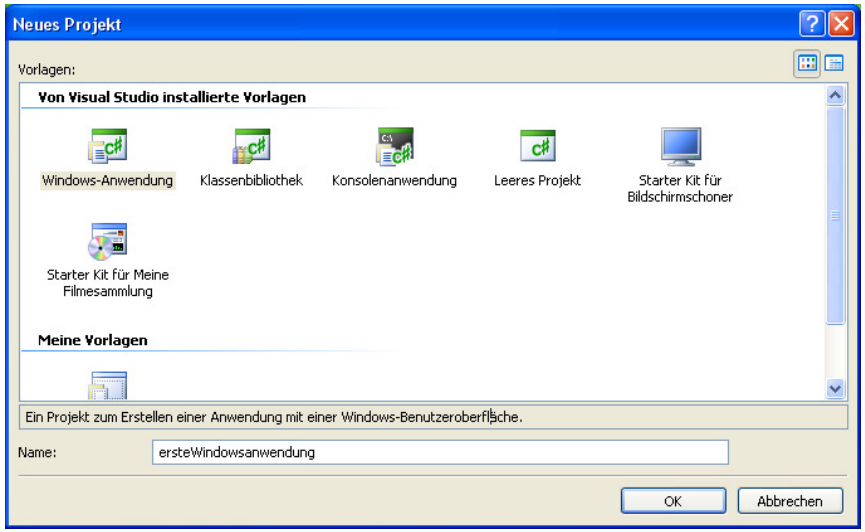
```
Geben Sie den monatlichen Betrag ein:100

Monat: 1        Saldo: 915.0  Gesamtbetrag: 100.0
Monat: 2        Saldo: 828.725        Gesamtbetrag: 200.0
Monat: 3        Saldo: 741.155875     Gesamtbetrag: 300.0
Monat: 4        Saldo: 652.273213125  Gesamtbetrag: 400.0
Monat: 5        Saldo: 562.057311321875       Gesamtbetrag: 500.0
Monat: 6        Saldo: 470.4881709917031      Gesamtbetrag: 600.0
Monat: 7        Saldo: 377.54549355657866     Gesamtbetrag: 700.0
Monat: 8        Saldo: 283.20867595992735     Gesamtbetrag: 800.0
Monat: 9        Saldo: 187.4568060993263      Gesamtbetrag: 900.0
Monat: 10       Saldo: 90.26865819081618      Gesamtbetrag: 1000.0
Monat: 11       Saldo: -8.377311936321576     Gesamtbetrag: 1100.0
```

Berechnen Sie die monatlichen Zinsen auf Grund des Saldos. Berechnen Sie dann den neuen Saldo in dem Sie die Zinsen addieren und den zurückgezahlten Betrag subtrahieren.

Verbessertes Programm: Lassen Sie das Programm nach dem Anfangssaldo, dem monatlichen Zinssatz und dem Rückzahlungsbetrag fragen. Wenn der Saldo unter dem Rückzahlungsbetrag liegt, gibt das Programm die letzte Rate aus, die den Saldo exakt auf 0 bringt.

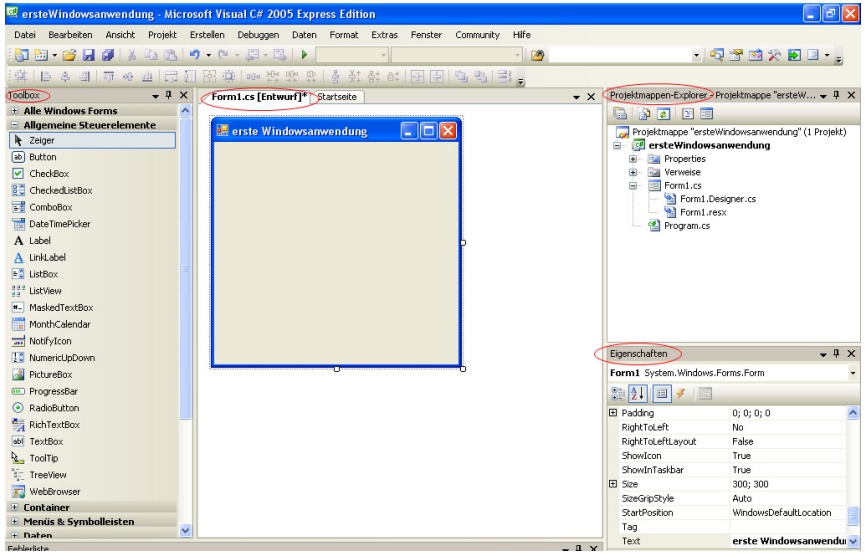
Erste Windowsanwendung

- Datei → „Neues Projekt“ auswählen
- Dann erscheint ein Fenster (siehe Abbildung 1):
  - Windows-Anwendungen auswählen
  - Projektname vergeben und nicht bei „WindowsApplication1“ belassen → Vorschlag „ersteWindowsanwendung“
  - OK-Button anklicken



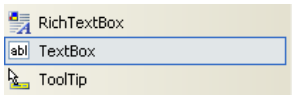
(Abbildung 1)

- Anschließend erscheint ein Windowsfenster (siehe Abbildung 2) mit vier Bereichen:
  - Im linken Bereich: Toolbox → Enthält alle grafischen Objekte, die man einbauen kann.
  - In der Mitte: „Form1.cs [Entwurf]“ → Das enthält ein Fenster „Form1“.
  - Im rechten Bereich oben: „Projektmappen-Explorer“ → Hier sieht man die Struktur des Projekts und die einzelnen Programmdateien.
  - Im rechten Bereich unten: „Eigenschaften“ → Enthält die Eigenschaften des markierten Objekts im Fenster „Form1“.
    - Markieren Sie das Fenster durch die Wahl mit der Maus: Gehen Sie mit der Maus in das Fenster Eigenschaften in die Zeile „Text“ und ändern den Fensterüberschrift, z.B. geben ein „erste Windowsanwendung“.



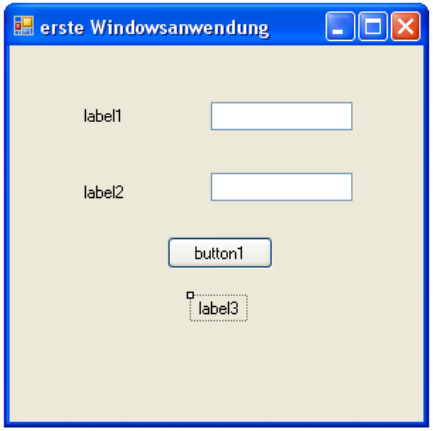
(Abbildung 2)

- Markieren Sie in der Toolbox die Textbox, dadurch wird ein blauer Rahmen um Textbox gelegt.



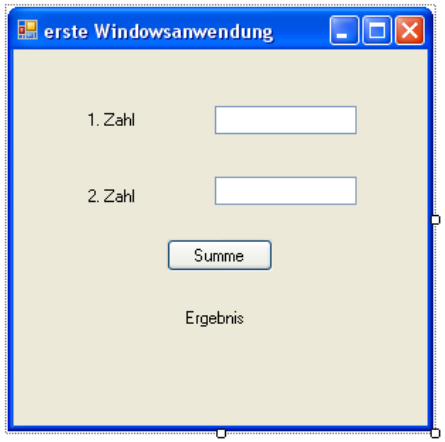
- Anschließend im Fenster „erste Windowsanwendung“ eine Stelle aussuchen, wo man die Textbox platzieren möchte

- Mit der linken Maustaste die Stelle wählen und dann wird dort eine Textbox platziert.
- Sie können später auch die Textbox verschieben, indem Sie das Feld mit der Maus auswählen.
- Platzieren Sie noch weitere Textboxen, Buttons und Labels wie in der Abbildung 3.



(Abbildung 3)

- Nennen Sie die Elemente um, in dem Sie die Felder anklicken und bei den Eigenschaften in der Zeile „Text“ den Namen wie in der Abbildung 4 dargestellt ändern.



(Abbildung 4)

- Passen Sie die Namen der Variablen, die auf diese Felder zugreifen:
  - Label mit dem Text „1.Zahl“ soll den Variablennamen „labelZahl1“ bekommen. Sie markieren das Label „1.Zahl“, gehen auf die Eigenschaften in die Zeile „Name“ und geben „labelZahl1“ ein.
  - Die Variablennamen der Textboxen, der Labels und der Box ändern:
    - Label mit dem Text „2.Zahl“ → „labelZahl2“
    - Label mit dem Text „Ergebnis“ → „labelErgebnis“
    - „textBox1“ → „textBoxZahl1“
    - „textBox2“ → „textBoxZahl2“
    - „button1“ → „buttonSumme“

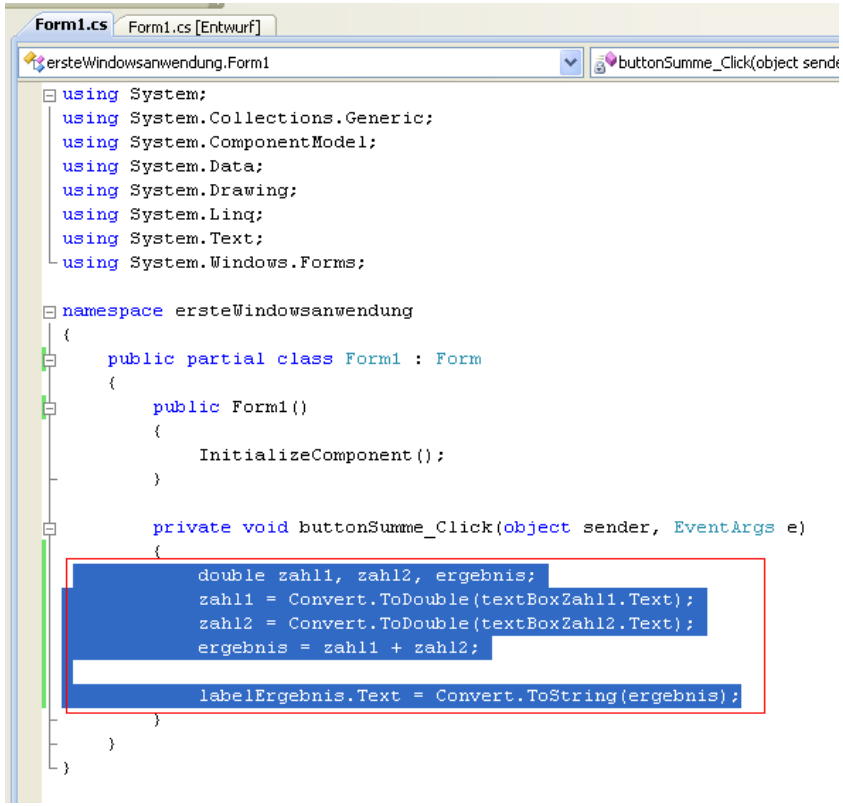
Damit auch eine Aktion durchgeführt wird, wenn man auf den Button klickt, muss noch programmiert werden.

Klicken Sie mit der Maus zweimal auf den Button „Summe“, dann sollte ein neues Registerblatt „Form.cs“ erscheinen. In der Datei „Form.cs“ wurde automatisch schon Quellcode erzeugt, man muss es nur ergänzen.

Es wurde der Rahmen für die Methode „buttonSumme_Click“ automatisch erzeugt. Den markierten Quellcode wie in der Abbildung 5 muss man selbst programmieren.

Im Folgenden wird die Bedeutung des Programms erklärt:

```
private void buttonSumme_Click(object sender, EventArgs e)
{
    // Deklariere die Variablen zahl1, zahl2 und ergebnis als
    // double (Dezimalzahl)
    double zahl1, zahl2, ergebnis;

    // Nimm den Text im Textbox "textboxZahl1" und konvertiere es in
    // ein double und speichere den Wert in die Variable zahl1
    zahl1 = Convert.ToDouble(textBoxZahl1.Text);

    // Nimm den Text im Textbox "textboxZahl2" und konvertiere es in
    // ein double und speichere den Wert in die Variable zahl2
    zahl2 = Convert.ToDouble(textBoxZahl2.Text);

    // addiere die Werte in den beiden VAriablen zahl1 und zahl2
    // und speichere das Ergebnis in die Variable ergebnis
    ergebnis = zahl1 + zahl2;

    // Konvertieren den Wert in der Variable "ergebnis" in
    // String (also in Text) und schreibe dies in das Textfeld
    // des Labels "labelErgebnis"
    labelErgebnis.Text = Convert.ToString(ergebnis);
}
```

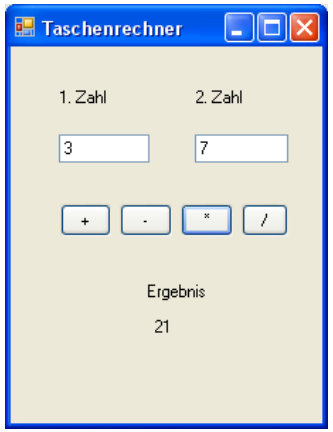



(Abbildung 5)

Programmieraufgaben(IX): Taschenrechner

Programmieren Sie in C# einen einfachen Taschenrechner.
Der Taschenrechner sollte mindestens die einfachen Grundrechenarten (Addition, Subtraktion, Multiplikation und Division) enthalten.

Der Taschenrechner könnte wie folgt aussehen:



Literatur

- C# von Eric Gunnerson: http://openbook.galileocomputing.de/visual_csharp_2010/
- Visual C# 2010 von Andreas Kühnel: http://openbook.galileocomputing.de/csharp/
-

Indexverzeichnis

Arithmetische Operatoren 11

Bedingte Anweisung 16

C#-Projekt 4

Datentyp 7

Datentyp Bool

Scheckgebühr.....39

Deklaration 8

elementare Datentypen 7

if-Anweisung 16

Initialisierung.....8

Logische Operatoren 14

Negationsoperator 14

Operatoren

Additionnsoperator 11

Dekrementoperator 11

Divisionsoperator.....11

Inkrementoperator 11

Modulo-Operator 11

Multiplikationsoperator..... 11

Restwertoperator 11

Subtraktionsoperator 11

Vorzeichenoperator..... 11

Schleifen 22

do-while-Schleife 22

for-Schleife..... 26

while-Schleife 22

Zählergesteuerte Schleife..... 26

Und-Operator..... 14

Variablen..... 7

Vergleichsoperatoren..... 14

Verzweigung..... 16

Zuweisungsoperator..... 8