

# Modul 319

## Block 02

### - Kontrollstrukturen -



#### Dokumentenversion:

V1.1	Erstellung durch Roland Bucher
V1.2	Integration der Übungen aus dem Stützkurs

#### Voraussetzung:

Dieser Block setzt den Block 1 des Moduls 319 voraus.

#### Aufbau dieses Blocks:

Der Block stellt im ersten Kapitel in Form einer Zusammenfassung die wichtigsten Elemente von Verzweigungen (Auswahl) dar.

Im zweiten Kapitel werden die wichtigsten Elemente von Schleifen (Iterationen oder Wiederholungen) zusammengefasst. Das letzte Kapitel des Dokumentes enthält Übungen im Umfang von über 6 Lektionen zu diesen Themen.

#### Ziele dieses Blocks:

- Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen
- Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von einseitigen, zweiseitigen und mehrstufigen Verzweigungen in c#- oder Java-Programmcode realisieren.
- Die Lernenden können c#- und Java-Programmcode mit kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) korrekt interpretieren und Fehler in existierenden Programmen erkennen.
- Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren.

#### Zeitaufwand:

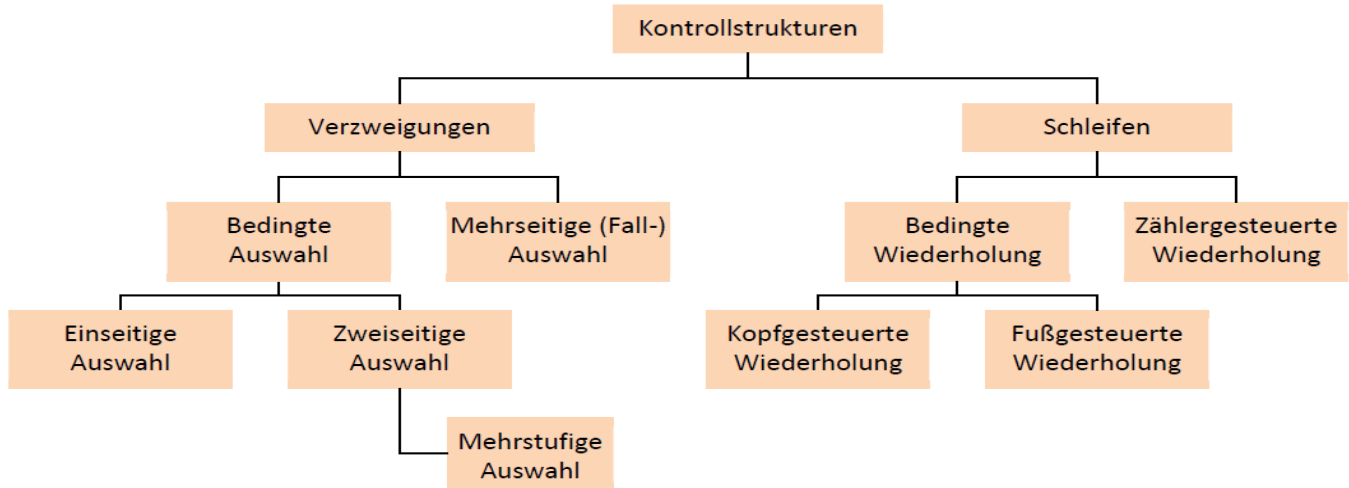
Für die Abarbeitung dieses Blockes sind im Unterricht 8 Lektionen vorgesehen. Der Rest ist als Hausaufgabe zu lösen.

#### Inhaltsverzeichnis:

1	Übersicht über Kontrollstrukturen .....	2
2	Verzweigungen.....	2
2.1	Einseitige Auswahl ( if ) .....	2
2.2	Zweiseitige Auswahl ( if-else ) .....	2
2.3	Auswahlen verschachteln.....	3
2.4	Mehrstufige Auswahl (Fallauswahl) .....	4
3	Schleifen (Wiederholungen) .....	5
3.1	Kopfgesteuerte Schleife (while).....	6
3.2	Fussgesteuerte Schleife (do-while).....	7
3.3	Zählergesteuerte Wiederholung (for).....	8
3.4	break und continue Anweisungen in Schleifen .....	9
4	Übungen (110').....	10

# 1 Übersicht über Kontrollstrukturen

Kontrollstrukturen bestehen aus zwei Kategorien: Verzweigungen (z.B. if-then-else) und Schleifen (z.B. while). Bei Verzweigungen geht es prinzipiell darum, ob eine Anweisung oder ein Block von Anweisungen ausgeführt werden soll oder nicht. Bei Schleifen geht es darum, dass eine Anweisung oder ein Block von Anweisungen unter Umständen mehrmals ausgeführt werden sollen.



## 2 Verzweigungen

(Ein Auszug aus ihrem Herdt-Buch C# - Grundlagen der Programmierung mit VisualStudio 2019, Kap. 7.2 bis 7.5)

### 2.1 Einseitige Auswahl ( if )

Bei vielen Problemstellungen ist die Verarbeitung von Anweisungen von Bedingungen abhängig. Nur wenn die Bedingung erfüllt ist, wird die betreffende Anweisung (bzw. der Anweisungsblock) ausgeführt. Anderenfalls wird die Anweisung übersprungen.

- ✓ Die einseitige Auswahl beginnt mit dem Schlüsselwort `if`.
- ✓ Dahinter wird – eingeschlossen in runde Klammern `()` – eine Bedingung (condition) formuliert. Die Bedingung ist ein Ausdruck (expression) und liefert als Ergebnis einen Wert vom Typ `bool` zurück.
- ✓ Anschließend folgt die Anweisung ①, die ausgeführt wird, wenn die Auswertung der Bedingung den Wert `true` (wahr) ergibt.
- ✓ Die Anweisung ① kann aus einer einzelnen Anweisung oder einem Anweisungsblock in geschweiften Klammern `{ }` bestehen.
- ✓ Liefert die Bedingung den Wert `false` (falsch), wird die Anweisung ① bzw. der Anweisungsblock übersprungen.

```
if (condition)
    statement; ①
```

### 2.2 Zweiseitige Auswahl ( if-else )

Im Gegensatz zur einseitigen Auswahl hat die zweiseitige Auswahl noch am Schluss einen `else`-Teil. Während die einseitigen Auswahl nur entscheiden kann, ob eine Anweisung oder ein Block von Anweisungen ausgeführt werden soll oder nicht, kann die zweiseitige Auswahl entscheiden, welcher der beiden Teile (if oder else) ausgeführt werden soll.

- ✓ Nach dem Schlüsselwort `if` steht in runde Klammern `()` eingeschlossen die Bedingung ①.
- ✓ Anschließend folgt die Anweisung ② oder der Anweisungsblock, die bzw. der ausgeführt werden soll, wenn die Bedingung erfüllt ist.
- ✓ Nach `else` schließt sich die Anweisung ③ oder der Anweisungsblock an, die bzw. der ausgeführt wird, wenn die Bedingung in der `if`-Anweisung nicht zutrifft (Ausdruck liefert `false`). Die Anweisung ② bleibt unberücksichtigt.

```
if (condition) ①
{
    statement1 ②
}
else
{
    statement2 ③
}
```

## 2.3 Auswahlen verschachteln

Eine mehrstufige Auswahl erreichen Sie in C#, indem Sie mehrere `if`- bzw. `if-else`-Anweisungen schachteln. Dazu schreiben Sie innerhalb eines Anweisungsblocks einer `if`- bzw. `if-else`-Anweisung eine weitere `if`- bzw. `if-else`-Anweisung.

Zu welcher <code>if</code> -Anweisung gehört die <code>else</code> -Alternative?	
<pre> if (condition) {     if (condition2) ②         statement1     else ①         statement2         </pre>	<pre> if (condition) ④ {     if (condition2)         statement1 } else ③     statement2         </pre>
Die mit <code>else</code> eingeleitete Alternative ① gehört zur jeweils vorherigen <code>if</code> -Anweisung ②.	Soll der <code>else</code> -Zweig ③ zu einer weiter entfernten <code>if</code> -Anweisung ④ gehören, müssen Sie dies durch die Verwendung von Anweisungsblöcken ⑤ sicherstellen.

## 2.4 Vergleichs- und Verknüpfungsmöglichkeiten bei der Formulierung von Bedingungen

Eine Bedingung (Condition) muss gesamthaft immer auf `true` oder `false` auswertbar sein. Dazu existieren verschiedene Vergleichsmöglichkeiten:

<code>==</code>	Überprüft zwei Ausdrücke auf Gleichheit (alle primitiven Datentypen)
<code>!=</code>	Überprüft zwei Ausdrücke auf Ungleichheit (alle primitiven Datentypen)
<code>&gt;</code>	Liefert <code>true</code> , wenn der erste Ausdruck größer als der zweite ist (alle außer <code>bool</code> )
<code>&lt;</code>	Liefert <code>true</code> , wenn der erste Operand kleiner als der zweite ist (alle außer <code>bool</code> )
<code>&gt;=</code>	Liefert <code>true</code> , wenn der erste Operand größer oder gleich dem zweiten ist (alle außer <code>bool</code> )
<code>&lt;=</code>	Liefert <code>true</code> , wenn der erste Ausdruck kleiner oder gleich dem zweiten ist (alle außer <code>bool</code> )

Nicht selten gibt es mehrere Bedingungen, die untereinander mit UND oder ODER verknüpft sind. Zum Beispiel: Die Variable `x` muss grösser als 10 UND kleiner als 20 sein:

```

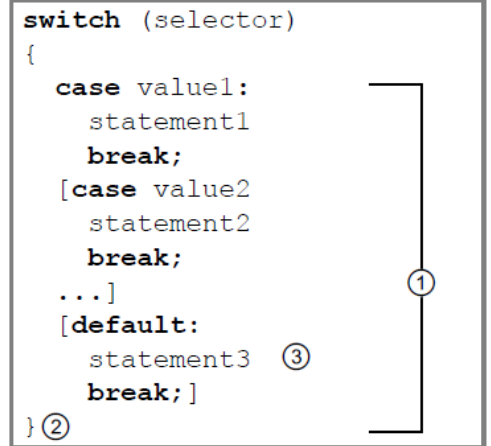
if (x>10 && x<20)
{
    ...
}
        
```

Operator	Bedeutung
<code>&amp;&amp;</code>	Bei booleschen Datentypen einsetzbar: Wie <code>&amp;</code> , jedoch wird der zweite Ausdruck nur ausgewertet, wenn der erste Ausdruck <code>true</code> liefert.
<code>  </code>	Bei booleschen Datentypen einsetzbar: Wie <code> </code> , jedoch wird der zweite Ausdruck nur ausgewertet, wenn der erste Ausdruck <code>false</code> liefert.

## 2.5 Mehrstufige Auswahl (Fallauswahl)

Bei `if`-Konstruktionen lassen sich immer nur zwei alternative Programmteile ausführen, da eine Bedingung als logischer Ausdruck nur die Werte `true` bzw. `false` ergeben kann. Bei einer mehrseitigen Auswahl testen Sie den Wert einer Variablen oder eines komplexen Ausdrucks. Diese Variable bzw. dieser Ausdruck wird als Selektor bezeichnet. In Abhängigkeit vom Wert des Selektors können fallweise (`case`) verschiedene Anweisungsblöcke ausgeführt werden. Die mehrseitige Auswahl wird daher auch Selektion genannt.

- ✓ Das Schlüsselwort `switch` leitet die mehrseitige Auswahl (Verzweigung) ein.
- ✓ Danach folgt der **Selektor**, der ein Ausdruck oder eine Variable sein kann. Der Wert des Selektors bestimmt, welche Anweisung als Nächstes ausgeführt wird. Der Selektor muss von einem ganzzahligen Datentyp oder vom Datentyp `string` sein.
- ✓ Auf den Selektor folgt ein Anweisungsblock ① mit sogenannten **case-Anweisungen**.
- ✓ Jede `case`-Anweisung beginnt mit dem Schlüsselwort `case`. Für jeden zu betrachtenden Wert, den der Selektor annehmen kann, wird eine `case`-Anweisung formuliert.



```

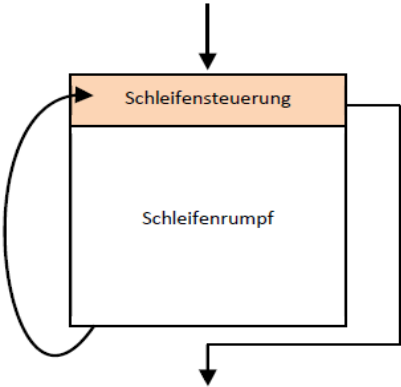
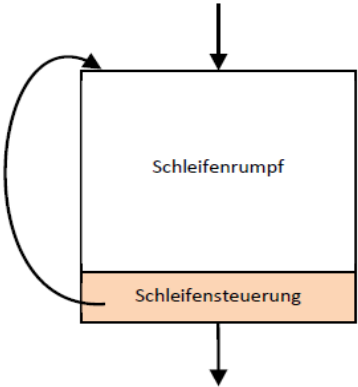
...
static void Main(string[] args)
{
    string name = "";
    Console.Write("Geben Sie einen Monatsnamen ein: ");
    name = Console.ReadLine();
    ① switch (name)
    {
    ②     case "Januar":
    ③     case "Februar":
    ④     case "März":
    ⑤         Console.WriteLine("1. Quartal");
    ⑥         break;
        case "April":
        ...
        default:
            Console.WriteLine("ungültiger Monatsname");
            break;
    }
}
    
```

- ① Die `string`-Variable `name`, die den eingegebenen Monatsnamen enthält, wird als Selektor genutzt.
- ② Die `case`-Anweisung mit dem Wert "Januar" enthält keine Anweisung (auch keine `break`-Anweisung). Der Fall wird automatisch zum nächsten Fall weitergeleitet. Man spricht auch von "Durchfallen zum nächsten Fall".
- ③ Auch der Fall "Februar" fällt durch (zum Fall "März" ④).
- ④ Der Fall "März" enthält eine Anweisung.
- ⑤ Diese Anweisung wird für alle drei Fälle ("Januar", "Februar" und "März") ausgeführt.
- ⑥ Die `break`-Anweisung beendet den `case`-Zweig und damit die `switch`-Anweisung.

### 3 Schleifen (Wiederholungen)

(Ein Auszug aus dem Herdt-Buch C# - Grundlagen der Programmierung mit VisualStudio 2019, Kap. 7.6 bis 7.10)

Eine Schleife besteht aus der Schleifensteuerung und dem Schleifenrumpf. Der Schleifenrumpf umfasst den Programmteil, der wiederholt werden soll. Die Schleifensteuerung legt fest, wie oft die Anweisungen im Schleifenrumpf wiederholt werden sollen bzw. nach welchen Kriterien entschieden wird, ob eine Wiederholung erfolgen soll. Es werden zwei Steuerungsarten unterschieden:

Kopfgesteuerte Schleife	Fußgesteuerte Schleife
	
<ul style="list-style-type: none"> <li>✓ Die Prüfung, ob die Anweisungen im Schleifenrumpf ausgeführt werden sollen, erfolgt gleich zu Beginn.</li> <li>✓ Ist das Kriterium erfüllt, wird der Schleifenrumpf durchlaufen und anschließend erfolgt eine erneute Prüfung.</li> <li>✓ Falls das Kriterium zu Beginn bereits nicht erfüllt ist, wird der Schleifenrumpf <b>gar nicht</b> ausgeführt.</li> </ul>	<ul style="list-style-type: none"> <li>✓ Zuerst werden die Anweisungen im Schleifenrumpf ausgeführt.</li> <li>✓ Dann erfolgt die Prüfung, ob ein weiterer Durchlauf erfolgen soll.</li> <li>✓ Der Schleifenrumpf wird also immer <b>mindestens einmal</b> ausgeführt.</li> </ul>

### 3.1 Kopfgesteuerte Schleife (while)

Bei der `while`-Anweisung handelt es sich um eine kopfgesteuerte Schleifenstruktur. Die Ausführung der Anweisungen im Schleifenrumpf ist von der Gültigkeit einer Bedingung abhängig, die gleich zu Beginn der Anweisung überprüft wird. Solange (`while`) die Bedingung erfüllt ist, werden die folgenden Anweisungen (Schleifenrumpf) ausgeführt. Anschließend folgt der nächste Durchlauf der Schleife und die Bedingung wird erneut geprüft. Ist die Bedingung nicht erfüllt, wird das Programm hinter der `while`-Anweisung fortgesetzt.

- ✓ Das Schlüsselwort `while` leitet die Anweisung ein.
- ✓ Es folgt in runden Klammern `()` die Formulierung der Bedingung, die einen Wert vom Typ `bool` liefern muss.
- ✓ Anschließend folgt der Schleifenrumpf mit der Anweisung oder einem Anweisungsblock (`statement`).
- ✓ Ist die Bedingung erfüllt (`true`), wird der Anweisungsblock ausgeführt. Anschließend erfolgt eine erneute Prüfung der Bedingung. Solange die Bedingung erfüllt ist, wird die Ausführung der Anweisung(en) wiederholt.
- ✓ Ist die Bedingung nicht erfüllt (`false`), wird der Schleifenkörper übersprungen und das Programm nach der `while`-Anweisung fortgesetzt.

```
while (condition)
    statement;
```

Beispiel: Das folgende Programm gibt die Zahlen 1, 3, 5, 7, 9 aus. Dazu wird beim Startwert 1 begonnen und in jedem Schleifendurchlauf der Wert 2 addiert, solange der Wert der Variablen `counter` kleiner als 10 ist.

```
static void Main(string[] args)
{
    ① int counter = 1;
    ② while (counter < 10)
    {
        ④ Console.WriteLine(counter); ③
        ⑤ counter += 2;
    }
}
```

- ① Der Startwert der Schleife wird mit 1 festgelegt.
- ② Die Bedingung der Schleife legt fest, dass der Schleifenrumpf ③ ausgeführt wird, solange der Wert der Variablen `counter` kleiner als 10 ist.
- ④ Der Wert der Variablen `counter` wird ausgegeben.
- ⑤ Die Zählervariable `counter` wird um 2 erhöht, anschließend wird wieder Schritt ② ausgeführt.



### 3.2 Fussgesteuerte Schleife (do-while)

Die `do-while`-Anweisung ist eine fussgesteuerte Schleifenstruktur. Die Bedingung wird erst nach dem Durchlauf des Schleifenrumpfes ausgewertet. Dadurch wird der Schleifenkörper mindestens einmal ausgeführt. Wie bei der kopfgesteuerten `while`-Anweisung wird der Schleifenrumpf ausgeführt, solange die Bedingung erfüllt ist.

- ✓ Das Schlüsselwort `do` leitet die Anweisung ein.
- ✓ Anschließend folgt der Schleifenrumpf mit der Anweisung oder mehreren Anweisungen innerhalb eines Anweisungsblocks in geschweiften Klammern (`statement`).
- ✓ Zur Einleitung der Schleifensteuerung folgt das Schlüsselwort `while`.
- ✓ Es folgt in runden Klammern `()` die Formulierung der Bedingung, die einen Wert vom Typ `bool` liefern muss.
- ✓ Abschließend folgt das Semikolon.
- ✓ Ist die Bedingung erfüllt (`true`), wird der Schleifenrumpf erneut durchlaufen. Ist die Bedingung nicht erfüllt (`false`), wird die `do-while`-Anweisung beendet und das Programm fortgesetzt.

```
do
    statement;
while (condition);
```

Beispiel: Das folgende Programm berechnet die Anzahl der Jahre, die benötigt werden, um – beginnend mit einem Startkapital von 1'000 EUR – nur über die Zinsen (4,5 %) einen Endbetrag von 10'000 EUR zu erreichen.

```
static void Main(string[] args)
{
    ① double presentValue = 1000.0;
    L double futureValue = 10000.0;
    ② const double interestRate = 4.5;
    ③ int year = 0;
    do
    {
    ④     presentValue *= 1.0 + interestRate / 100; // Startwert um Zinsen erhöhen
    L     year++;                                // Jahr um 1 erhöhen
                                                // Alternative: year++;
    }
    ⑤ while (presentValue < futureValue);
    //solange das gewünschte Kapital noch nicht erreicht ist
    ⑥ Console.WriteLine("Laufzeit in Jahren: " + year);
}
```

- ① Die Variablen enthalten die Werte für das Startkapital (`presentValue`) und das zu erreichende Zielkapital (`futureValue`). Da für die Zinsberechnung Kommastellen notwendig sind, werden Gleitkommazahlen (`double`) verwendet.
- ② Der Zinssatz wird mit 4,5 % als Konstante festgelegt.
- ③ Die Variable `year` enthält als Zähler die Anzahl der Jahre, die vergehen, bis das gewünschte Kapital erreicht ist.
- ④ Im Schleifenrumpf werden die Zinsen berechnet und zum aktuellen Kapital addiert. Der Jahreszähler (`year`) wird jeweils um 1 erhöht.
- ⑤ Solange das gewünschte Kapital nicht erreicht wurde, wird der Schleifenkörper erneut ausgeführt.
- ⑥ Ist das Kapital erreicht, wird die Anzahl der Jahre, die zum Erreichen des Endbetrags notwendig sind, ausgegeben.

### 3.3 Zählergesteuerte Wiederholung (for)

Wenn bereits zu Beginn der Schleife genau bekannt ist, wie oft der Schleifenrumpf (eine Anweisung oder ein Anweisungsblock) wiederholt werden soll, ist die Verwendung der for-Schleife die kompakteste Form.

- ✓ Das Schlüsselwort `for` leitet die zählergesteuerte Wiederholung ein.
 

```
for ([type] counter = start; condition; nextStatement)
    statement;
```
- ✓ Anschließend folgt in runden Klammern `()` die Schleifensteuerung. Diese beginnt mit der Festlegung der Zählervariablen (`counter`) auf den Startwert (`start`). Die Zählervariable kann auch hier deklariert werden und ist dann nur in dem Schleifenrumpf gültig. Die Zählervariable muss einen numerischen Datentyp besitzen.
- ✓ Abgetrennt durch ein Semikolon folgt die Bedingung (`condition`), die erfüllt sein muss, damit die Schleife erneut ausgeführt wird.
- ✓ Nach einem weiteren Semikolon folgt der Aktualisierungsteil (`nextStatement`). Als Anweisung beschreiben Sie hier beispielsweise, wie der Wert der Zählvariablen bei jedem Schleifendurchlauf verändert wird (z. B. Inkrementieren oder Dekrementieren). Die im Aktualisierungsteil aufgeführte Anweisung wird für jeden Schleifendurchlauf nur einmal ausgeführt.
- ✓ Für jeden Schleifendurchlauf wird die Anweisung bzw. der Anweisungsblock (`statement`) ausgeführt.

Beispiel: Das folgende Programm berechnet die Fakultät einer natürlichen Zahl im Bereich zwischen 1 und 15. Zum Beispiel ist die Fakultät der Zahl 4 gleich 24 (Berechnung:  $4! = 1 * 2 * 3 * 4$ ).

```
static void Main(string[] args)
{
    ① int result = 1;
    ② int number = 0;
    Console.WriteLine("Geben Sie eine Zahl zwischen 1 und 15 ein: ");
    string txt = Console.ReadLine();
    ③ number = Convert.ToInt32(txt);
    ④ if (number >= 1 && number <= 15)
    {
        ⑤ for (int i = 1; i <= number; i++)
        {
            ⑥ result *= i;
        }
        ⑦ Console.WriteLine("{0}! = {1}", number, result);
    }
    else
    {
        ⑧ Console.WriteLine("Die Zahl liegt nicht im gültigen Bereich");
    }
}
```

- ① Die Variable `result` speichert die berechnete Fakultät.
- ② Die Variable `number` speichert den Zahlenwert, für den die Fakultät berechnet werden soll.
- ③ Die übergebene Textzeile wird in eine Zahl vom Datentyp `int` umgewandelt und der Variablen `number` zugewiesen.
- ④ Für Zahlen kleiner als 1, für die die Fakultät nicht berechnet werden kann, und für Zahlen größer als 15 erfolgt eine entsprechende Ausgabe ⑧ und das Programm wird beendet.
- ⑤ Innerhalb der `for`-Anweisung wird die Variable `i` definiert und mit dem Wert 1 initialisiert. Die Schleife soll ausgeführt werden, solange der Zähler `i` kleiner oder gleich dem Wert `number` ist. Die Variable `i` wird nach jedem Schleifendurchlauf inkrementiert (um den Wert 1 erhöht).
- ⑥ Die Fakultät wird berechnet. Durch die Wiederholungen des Schleifenrumpfs entsteht eine Formel der Art: `result = 1 * 2 * 3 * ... * number`
- ⑦ Das Ergebnis der Berechnung wird ausgegeben.



### 3.4 break und continue Anweisungen in Schleifen

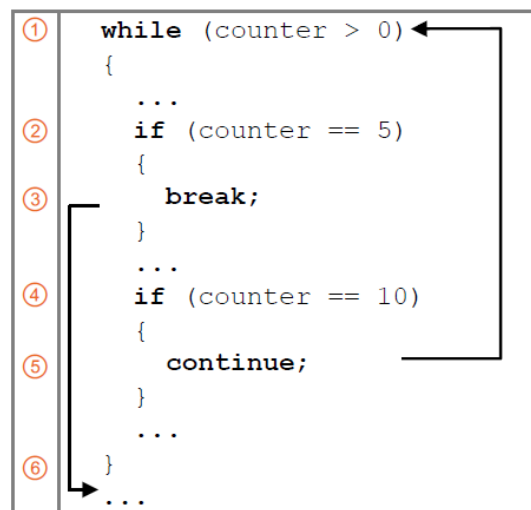
Zusätzlich zur Schleifensteuerung haben Sie mit einer `break`- bzw. einer `continue`-Anweisung die Möglichkeit, den Ablauf des Programms zu beeinflussen.

- ✓ Wie in der `switch`-Anweisung kann die `break`-Anweisung auch innerhalb von Schleifen angewendet werden, um die jeweilige Kontrollstruktur zu beenden.
- ✓ Die `continue`-Anweisung können Sie innerhalb von Schleifen einsetzen. Bei einer bedingten Wiederholung wird sofort die Testbedingung neu ausgewertet. Bei der zählergesteuerten Wiederholung (`for`) wird sofort der nächste Zählerwert verwendet und danach die Testbedingung neu ausgewertet.

Bei geschachtelten Kontrollstrukturen wirkt sich der Aufruf von `continue` bzw. `break` nur für die jeweils innere Schleife aus.

Beispiel:

- ① Hier beginnt eine `while`-Anweisung.
- ② Innerhalb der `while`-Anweisung wird eine Bedingung durch eine `if`-Anweisung geprüft.
- ③ Wenn die Bedingung bei ② erfüllt ist, wird die `while`-Anweisung durch die `break`-Anweisung verlassen. Die Anweisungen nach der `while`-Anweisung ⑥ werden als Nächstes ausgeführt.
- ④ Es wird eine weitere Bedingung durch eine `if`-Anweisung geprüft.
- ⑤ Wenn diese Bedingung erfüllt ist, wird die `continue`-Anweisung ausgeführt. Die Schleifenprüfung beginnt wieder bei ①. Die Anweisungen nach der `continue`-Anweisung werden in diesem Schleifendurchlauf nicht mehr ausgeführt.



## 4 Übungen (260' / 6 Lektionen)

### Aufgabe 01 (Programmcode mit einer Verzweigungen interpretieren können)

**Lernziel:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen

**Zeit:** 10'

**Aufgabe:** Was gibt das Programm aus, wenn folgende Werte eingegeben würden?

a) Zahl 1 = 6, Zahl 2 = 3

b) Zahl 1 = 7, Zahl 2 = 7

```
static void Main(string[] args)
{
    Console.Write("Zahl 1:");
    int zahl1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Zahl 2:");
    int zahl2 = Convert.ToInt32(Console.ReadLine());
    if (zahl2 < zahl1 && zahl1 > 5)
    {
        int temp = zahl1;
        zahl1 = zahl2;
        zahl2 = temp;
    }
    else
    {
        if (zahl1 == zahl2)
            zahl1 = 5;
        zahl2 = 8;
    }
    Console.WriteLine("Ausgabe 1 = " + Convert.ToString(zahl1));
    Console.WriteLine("Ausgabe 2 = " + Convert.ToString(zahl2));
    Console.ReadLine();
}
```

Ihre Lösungen:	
Aufgabe a	Aufgabe b
Zahl 1: 6	Zahl 1: 7
Zahl 2: 3	Zahl 2: 7
.....	.....
.....	.....
.....	.....
.....	.....
.....	.....

## Aufgabe 02 (Programmcode mit mehreren Verweigungen interpretieren können)

**Lernziel:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen

**Zeit:** 10'

**Aufgabe:** Was gibt das Programm aus, wenn folgende Noten eingegeben würden?

a) Note 1 = 4.2, Note 2 = 3.5

b) Note 1 = 5.2, Note 2 = 5.8

c) Note 1 = 3.8, Note 2 = 3.3

```
static void Main(string[] args)
{
    int anzahl = 0;
    double note1 = 0.0, note2 = 0.0;
    Console.WriteLine("Note 1:");
    note1 = Convert.ToDouble(Console.ReadLine());
    anzahl++;
    Console.WriteLine("Note 2:");
    note2 = Convert.ToDouble(Console.ReadLine());
    anzahl++;
    double schnitt = (note1 + note2) / anzahl;
    if (schnitt >= 4)
    {
        Console.WriteLine("*****");
        int durchschnitt = (int)(schnitt * 2);
        durchschnitt = (durchschnitt + 1) / 2;
        if (durchschnitt == 4)
            Console.WriteLine("Typ 2");
        else
        {
            if (durchschnitt == 5)
                Console.WriteLine("Typ 3");
            else
                Console.WriteLine("Typ 4");
        }
    }
    else
    {
        Console.WriteLine("-----");
        if (note1 >= 4 || note2 >= 4)
            Console.WriteLine("Typ 1");
        else
            Console.WriteLine("Typ 0");
    }
    Console.ReadLine();
}
```

Ihre Lösungen:		
Aufgabe a	Aufgabe b	Aufgabe c
Note 1: 4.2 Note 2: 3.5	Note 1: 5.2 Note 2: 5.8	Note 1: 3.8 Note 2: 3.3
.....	.....	.....
.....	.....	.....
.....	.....	.....

### Aufgabe 03 (Fehler im Programmcode erkennen und korrigieren können)

**Lernziel:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen

**Zeit:** 10'

**Aufgabe:** Das folgende Programm sollte drei Werte (Wert 1, Wert 2 und Wert 3) vom Benutzer entgegennehmen und den grössten der drei Werte wieder ausgeben. Leider haben sich zwei Fehler eingeschlichen. Erklären sie am Schluss bei welcher Zeile (Zeilennummer) es warum einen Compilerfehler gibt und versuchen sie ihn zu korrigieren.

```
01 static void Main(string[] args)
02 {
03     int anzahl = 1;
04     double wert1 = 0, wert2 = 0, wert3=0, max= 0;
05     Console.Write("Wert " + Convert.ToString(anzahl++) );
06     wert1 = Convert.ToDouble(Console.ReadLine());
07     Console.Write("Wert " + Convert.ToString(anzahl++));
08     wert2 = Convert.ToDouble(Console.ReadLine());
09     Console.Write("Wert " + Convert.ToString(anzahl++));
10     wert3 = Convert.ToDouble(Console.ReadLine());
11     if (wert1 > wert2 && > wert3)
12         max = wert1;
13     else
14     {
15         if (wert2 > wert3)
16             max = wert2;
17         else (wert3 > wert2)
18             max = wert3;
19     }
20     Console.WriteLine("Der grösste Wert ist:" + Convert.ToString(max));
21     Console.ReadLine();
22 }
```

**Kompilerfehler 1** bei Zeile[n] Nummer: .....

**Grund:** .....

**Korrektur:** .....

**Kompilerfehler 2** bei Zeile[n] Nummer: .....

**Grund:** .....

**Korrektur:** .....

#### Aufgabe 04 (Gegebene Aufgabenstellung in Programmcode umsetzen können)

**Lernziel:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen

**Zeit:** 25'

**Aufgabe:** Beim Ringen wird in verschiedenen Gewichtsklassen gerungen.

Gewichtsklasse	Männer	Frauen
Fliegengewicht	–55 kg	–48 kg
Leichtgewicht	–66 kg	–55 kg
Mittelgewicht	–84 kg	–63 kg
Schwergewicht	–120 kg	–72 kg



Schreiben Sie ein Programm, das den Benutzer zur Eingabe der beiden Informationen Geschlecht und Gewicht auffordert.

Im Verarbeitungsteil ermittelt ihre App mit Hilfe von Verzweigungen in welcher Kategorie gerungen werden kann. Die ermittelte Information wird danach dem Benutzer ausgegeben.

Gibt der Benutzer zum Beispiel folgende Werte ein:

Geschlecht [m/w] : w

Gewicht [in Kg] : 75

Ausgabe = Gewichtsklasse: Mittelgewicht

#### Aufgabe 05 (Jahreszeiten mit Fallauswahl)

**Thema:** Eingabe, Verarbeitung (mit Fallauswahl), Ausgabe

**Lernziele:** Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von mehrstufigen Verzweigungen in c#- oder Java-Programmcode realisieren.

**Zeit:** 20'

**Aufgabe:** Schreiben Sie ein Programm, das den Benutzer zur Eingabe eines Monats auffordert (Zahl von 1 bis 12).

Im Verarbeitungsteil ermittelt ihre App mit Hilfe einer mehrstufigen Verzweigung (switch -case) ob es sich um einen Frühlings, Sommer-, Herbst- oder Wintermonat handelt. Die ermittelte Information wird danach dem Benutzer ausgegeben.

Gibt der Benutzer zum Beispiel folgende Werte ein:

Monat : 4

Erfolgt die Ausgabe:

Der Monat 4 ist ein Frühlingsmonat

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

```
static void Main(string[] args)
{
    Console.WriteLine("Jahreszeiten");
    Console.WriteLine("*****");

    Console.ReadLine();
}
```

### Aufgabe 06 (Gegebene Aufgabenstellung in Programmcode umsetzen können)

**Lernziel:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen

**Zeit:** 25'

**Aufgabe:** Untenstehend existiert eine Anleitung in Pseudocode/Textform, anhand der man zu jedem beliebigen Datum den zugehörigen Wochentag (z.B. Montag) ermitteln kann. Schreiben sie ein Programm, das vom Benutzer die drei ganzzahlige Werte t (Tag), m (Monat) und j (Jahr) entgegennimmt und eine Aussage in Textform macht, um welchen Wochentag es sich handelt.

**Testdaten:** 25.10.2021 → Montag ,      24.12.1980 → Mittwoch,      07.04.2037 → Dienstag  
Weitere Testdaten mit dem Windowskalender überprüfen

#### Anleitung in Pseudocode / Textform:

- 1) Erstellen Sie drei Variablen ( t, m, j ) vom Typ int und lesen Sie die Werte ein
- 2) Berechnen Sie den Wochentag h nach folgendem Algorithmus:
  - Falls  $m \leq 2$  ist, erhöhe m um 10 und erniedrige j um 1, andernfalls erniedrige m um 2.
  - Berechne die ganzzahligen Werte  $c = j/100$  und  $j = j \text{ Modulo } 100$  (Modulo → %)
  - Berechne den ganzzahligen Wert  
$$h = (((26*m-2)/10)+t+j/4+c/4-2*c) \text{ Modulo } 7$$
  - Falls h kleiner 0 ist, erhöhe h um 7
  - Anschliessend hat h einen Wert zwischen 0 und 6, wobei die Werte 0,1,...,6 den Tagen Sonntag, Montag, ....., Samstag entsprechen.
- 3) Geben Sie das Ergebnis in der Form "Der 24.12.2001 ist ein Montag" aus.



## Aufgabe 07 (Gegebene Aufgabenstellung in Programmcode umsetzen können)

**Lernziel:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen

**Zeit:** 60'

**Aufgabe:** Untenstehend existiert eine Anleitung in Pseudocode/Textform, anhand der man zu jedem beliebigen Jahr das Datum des Ostersonntags ermitteln kann. Ostern fällt immer auf den Sonntag nach dem ersten Vollmond im Frühling. Es hat also mit den Bewegungen von Erde, Sonne und Mond zu tun. Klingt komplex, ist es auch. Zum Glück hat der Mathematiker Carl Friedrich Gauss (1777 - 1855) dieses Problem bereits mathematisch für uns gelöst. Nachfolgend wird Ihnen textuell - gespickt mit Mathematik - erklärt, wie man exemplarisch für das Jahr 2019 berechnen kann, wann der Ostersonntag ist. Schreiben Sie ein Programm, das vom Benutzer eine Jahreszahl entgegennimmt und für das gewählte Jahr das Osterdatum ausgibt.



**Testdaten:** Testen Sie die Korrektheit Ihrer App mittels den folgenden Testwerten (**fett**= Osterparadoxon-Jahr):

<b>1974:</b> 14. April	2006: 16. April	2011: 24. April	2016: 27. März	2021: 04. April
1899: 02. April	2007: 08. April	2012: 08. April	2017: 16. April	2022: 17. April
1900: 15. April	2008: 23. März	2013: 31. März	2018: 01. April	2023: 09. April
1901: 07. April	2009: 12. April	2014: 20. April	<b>2019:</b> 21. April	2024: 31. März
<b>2000:</b> 23. April	2010: 04. April	2015: 05. April	2020: 12. April	<b>2038:</b> 25. April

### Anleitung in Pseudocode / Textform:

I) Die Zwischenwerte  $m$  und  $n$  bestimmen:

Zuerst müssen wir die Zwischenwerte  $m$  und  $n$  bestimmen, die je nach Jahreszahl variieren können. Z.B. ist im Jahr 2019  $m=24$  und  $n=5$ . Die nebenstehende Tabelle zeigt die Sollwerte von  $m$  und  $n$  je nach Jahreszahl. Kann man  $m$  und  $n$  auch berechnen? Klar:

Ab dem Jahr	$m$	$n$
1583	22	2
1700	23	3
1800	23	4
1900	24	5

II) Die Zwischenwerte  $m$  und  $n$  berechnen:

Step	Allgemein	Für das Jahr 2019
01	$m = (8 * (\text{Jahr}/100) + 13) / 25 - 2$	$m = (8 * (2019/100) + 13) / 25 - 2$ <b><math>m = (8 * 20 + 13) / 25 - 2 = 4</math></b>
02	$s = \text{Jahr}/100 - \text{Jahr}/400 - 2$	<b><math>s = 2019/100 - 2019/400 - 2 = 13</math></b>
03	$m = (15 + s - m) \% 30$	<b><math>m = (15 + s - m) \% 30 = (15 + 13 - 4) \% 30 = 24</math></b>
04	$n = (6 + s) \% 7$	<b><math>n = (6 + 13) \% 7 = 5</math></b>

III) Die Zwischenwerte  $d$  und  $e$  berechnen:

Step	Allgemein	Für das Jahr 2019
05	$a = \text{Jahr} \% 19$ $b = \text{Jahr} \% 4$ $c = \text{Jahr} \% 7$	$a = 2019 \% 19 = 5$ $b = 2019 \% 4 = 3$ $c = 2019 \% 7 = 3$
06	$d = (19 * a + m) \% 30$	<b><math>d = (19 * a + m) \% 30 = (19 * 5 + 24) \% 30 = 29</math></b>
07	Falls $d$ gleich 29 ist setze $d$ auf 28 andernfalls: falls $d=28$ und $a \geq 11$ ist, setze $d=27$	<b><math>d = 28</math></b>
08	$e = (2 * b + 4 * c + 6 * d + n) \% 7$	$e = (2 * b + 4 * c + 6 * d + n) \% 7$ <b><math>e = (2 * 3 + 4 * 3 + 6 * 28 + 5) \% 7 = 2</math></b>

IV) Nun können der Tag und der Monat bestimmt werden:

09	Ostertag = am $(22 + d + e)$ ten März	Ostertag = am $(22 + 28 + 2)$ ten März Ostertag = am 52. März
10	Falls der Ostertag $> 31$ ist, setze den Ostertag = Ostertag%31 und Ostermonat auf April	Ostertag = Ostertag%31 <b>Ostertag = 52%31 = 21</b> <b>Ostermonat = 4</b>

- Weitere Infos: <https://de.wikipedia.org/wiki/Osterparadoxon>

### Aufgabe 08 (Gegebene Aufgabenstellung in Programmcode umsetzen können)

**Lernziele:**

- Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen.
- Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 30'

**Aufgabe:** Mit Sicherheit kennen sie noch die Teilbarkeitsregeln aus dem Primarschulunterricht:

- Eine Zahl ist durch 3 teilbar, wenn die Quersumme durch 3 teilbar ist.
- Eine Zahl ist durch 9 teilbar, wenn die Quersumme durch 9 teilbar ist.
- etc.

Schreiben sie ein Programm, bei dem sie eine ganze Zahl eingeben können.

Für die eingegebene Zahl wird dann die Quersumme berechnet und eine Aussage zur Teilbarkeit durch 3 und 9 gemacht.

Am Ende wird gefragt, ob eine weitere Berechnung erwünscht ist. Falls Ja, wird der Bildschirminhalt gelöscht (Konsole.Clear() entspricht «cls») und das Programm erneut ausgeführt.

Beispiel: Zahl: 5989785  
Quersumme = 51  
Teilbar durch 3: Ja  
Teilbar durch 9: Nein

Möchten sie eine weitere Berechnung [ j / n ]: n

### Aufgabe 09 (Gegebene Aufgabenstellung in Programmcode umsetzen können)

**Lernziele:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen. Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 30'

**Aufgabe:** Schreiben Sie ein Programm, das dem Benutzer 6 » zufällige » Zahlen im Bereich von 1 bis 42 und eine »Glückszahl« im Bereich von 1 bis 6 ausgibt. Überprüfen sie, dass der Lotto-Tipp keine doppelten Zahlen enthält (also 12 soll z.B. nicht 2x vorkommen. Die Ausgabe muss noch nicht sortiert ausgegeben werden.



**Beispiele:** Ausgabe = 41, 29, 22, 36, 24, 5    Glückszahl = 3  
Ausgabe = 26, 12, 16, 25, 30, 18    Glückszahl = 5  
Ausgabe = 13, 28, 12, 23, 42, 19    Glückszahl = 1

## Aufgabe 10 (Gegebene Aufgabenstellung in Programmcode umsetzen können)

**Lernziele:** Die Lernenden können c#- und Java-Programmcode mit einseitigen, zweiseitigen und mehrstufigen Verzweigungen korrekt interpretieren und Fehler in existierenden Programmen erkennen. Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 60'

**Aufgabe:** Ihr Freund Bonifacius Pfiffikus erzählt ihnen von einer Theorie die er gehört habe. Die Theorie des Josefsrappen:

*„Geld, das Zinseszinsen trägt, wächst anfangs langsam; da aber durch den Zinseszinseffekt die Rate fortwährend wächst, wird sie nach einiger Zeit unvorstellbar schnell wachsen. Ein Rappen, ausgeliehen bei der Geburt von Jesus im Jahre 0 zu einem Zinssatz von 5 %, würde heute zu einer grösseren Summe herangewachsen sein, als 70 Milliarden Erden aus purem Gold entsprechen.“*

Sie können dies kaum glauben und möchten die Aussage mit einem zu erstellenden Programm überprüfen. Fragen Sie den Benutzer nach der Eingabe eines Start- und Zieljahres, nach einem Betrag und einem Zinssatz. Als Antwort liefert ihre App den Betrag in CHF und die Entprechung in Anzahl Erden aus purem Gold.



### Recherchen:

- Berechnen Sie das Volumen der Erde anhand des Erdradius. Obwohl die Erde nicht exakt eine Kugel ist, approximieren wir sie als eine Kugel. Die Formel um ein Kugelvolumen zu berechnen wie auch den Erdradius finden sie im Web.
- Recherchieren Sie, welches Volumen ein Kilogramm Gold hat und was dies kostet

### Tipp:

Die Zinseszinsen können sie auf zwei Arten berechnen. Mit Verwendung der Zinseszinsformel oder mit Verwendung einer Schleife die jedes Jahr die Jahreszinsen dazurechnet.

Mit der Zinseszinsformel wird angenommen, dass die Zinsen jährlich dem Anlagekapital zugeschlagen und dann ihrerseits verzinst werden. Dieser Zins auf Zinsen wird als Zinseszins bezeichnet und in der Zinseszinsformel berücksichtigt.

$$K_n = K_0 \cdot ((p / 100) + 1)^n$$

$K_n$ : Endkapital inkl. Zinsen nach n Jahren

$K_0$ : angelegtes Anfangskapital

p : Zinssatz in Prozent

n : Anzahl der Jahre

### Hintgergrundinfos: (Auszug aus Wikipedia):

Das Gedankenexperiment vom Josefspfennig geht zurück auf den britischen Moralphilosophen, Geistlichen und Ökonom Richard Price (1723-1791) und illustriert in der Zinsrechnung das im englischen Sprachraum als miracle of compound interest bekannte Wachstum eines über einen langen Zeitraum angelegten Vermögens durch Zinseszinsen.

### Aufgabe 11 (Ratespiel)

**Lernziele:** Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von Kontrollstrukturen (Schleifen und Verzweigungen) in c#- oder Java-Programmcode realisieren

**Zeit:** 60'

**Aufgabe:** Ihre Aufgabe ist ein Ratespiel zu programmieren. Der Computer generiert eine zufällige ganze Zahl zwischen 1 und 100. Sie haben fünf Rateversuche um die Zahl zu erraten. Nach jedem ihrer Versuche erhalten sie vom Computer ein Feedback, ob sie richtig geraten haben oder nicht. Das Programm gibt ihnen sogar einen Tipp, ob ihr Rateversuch zu hoch oder zu tief lag. Wenn es ihnen innert fünf Versuchen nicht gelingt, wird ihnen mitgeteilt, dass sie verloren haben. Am Ende eines Spiels werden sie gefragt, ob sie nocheinmal Raten möchten oder nicht.

```
Ihr 1er Rateversuch:50
Die geratene Zahl ist zu tief
Ihr 2er Rateversuch:75
Die geratene Zahl ist zu tief
Ihr 3er Rateversuch:89
Die geratene Zahl ist zu hoch
Ihr 4er Rateversuch:80
Die geratene Zahl ist zu hoch
Ihr 5er Rateversuch:77
Du hast gewonnen!
Moechtest du ein neues Spiel starten? [j/n]?
```

```
Ihr 1er Rateversuch:50
Die geratene Zahl ist zu hoch
Ihr 2er Rateversuch:30
Die geratene Zahl ist zu tief
Ihr 3er Rateversuch:40
Die geratene Zahl ist zu tief
Ihr 4er Rateversuch:42
Die geratene Zahl ist zu hoch
Ihr 5er Rateversuch:40
Die geratene Zahl ist zu tief
Du hast verloren!!
Die Zahl waere 41 gewesen!
Moechtest du ein neues Spiel starten? [j/n]?
```

## **Aufgabe 12 (Zahlensequenzen ausgeben)**

**Lernziele:** Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 30'

**Aufgabe:** Schreiben Sie ein Programm, das vom Benutzer eine Start- und eine Endzahl endgegennimmt und speichert. Ihre App gibt danach alle Zahlen von der Start- bis zur Endzahl, getrennt durch ein Komma und ein Leerschlag so aus, dass immer 10 Zahlen pro Zeile erscheinen. Beim Zeilenende um am Schluss sollen aber die Kommas nicht erscheinen.

Verwenden sie dazu den Schleifentyp `while(...) { ... }`

Gibt der Benutzer zum Beispiel folgende Werte ein:

Startzahl : 62

Endzahl : 87

Gibt das Programm als Resultat folgendes aus:

62, 63, 64, 65, 66, 67, 68, 69, 70, 71

72, 73, 74, 75, 76 ,77, 78, 79, 80, 81

82, 83, 84, 85, 86, 87

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

```
static void Main(string[] args)
{

    Console.ReadLine();
}
```

### Aufgabe 13 (Eine Dezimalzahl ins Binärsystem umrechnen)

**Lernziele:** Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 45'

**Aufgabe:** Schreiben Sie ein Programm, das eine Dezimalzahl zwischen 0 und 99'999 entgegennimmt und diese Zahl im Binärsystem ausgibt.

Gibt man zum Beispiel die Dezimalzahl 328 ein, gibt die App als Resultat 101001000 aus.

Dezimalzahl von 0 bis 99'999: 328  
Binaerzahl :101001000

**Tipp:** Mann kann dieses Problem auf versch. Arten lösen. In meiner Lösung bin ich so vorgegangen, dass ich die Dezimalzahl (hier z.B. 84) immer zuerst Modulo 2 rechne (ergibt die Binärziffer) und danach mit der ganzzahligen Division durch 2 die Dezimalzahl mindestens halbiere und solange weiterfahre bis die Dezimalzahl 0 ist.

84	%2	-->	0
84/2 --> 42	%2	-->	0
42/2 --> 21	%2	-->	1
21/2 --> 10	%2	-->	0
10/2 --> 5	%2	-->	1
5/2 --> 2	%2	-->	0
2/2 --> 1	%2	-->	1
84 -->	1010100		

**Ihre Lösung (Kopieren sie am Schluss ihre Lösung hier hinein):**

```
static void Main(string[] args)
{

    Console.ReadLine();

}
```



## Aufgabe 14 (ggT und kgV zweier Zahlen bestimmen)

**Thema:** Eingabe, Verarbeitung (mit Verzweigungen und Schleifen), Ausgabe

**Lernziel:** Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fussgesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 45'

**Aufgabe:** Bei der Bruchrechnung haben sie den Einsatz des ggT (grösster gemeinsamer Teiler) und des kgV (kleinste gemeinsame Vielfache) gelernt. Schreiben sie eine App, die den Benutzer auffordert zwei ganze Zahlen einzugeben. Berechnen sie danach für eingegebenen Zahlen den ggT und das kgV und geben sie diese Informationen dem Benutzer aus.

Der **ggT** ist die grösste Zahl, durch die sowohl A als auch B ohne Rest teilbar sind. Das **kgV** ist die kleinste Zahl, die sowohl ein Vielfaches der ersten Zahl als auch ein Vielfaches der zweiten Zahl ist.

Für die Berechnung des ggT sollen sie dabei den Algorithmus nach Euklid umsetzen. Dieser funktioniert wie folgt:

Man teilt die grössere durch die kleinere Zahl. Geht die Division auf, ist der Divisor der ggT. Geht die Division nicht auf, bleibt ein Rest. Dieser Rest ist der neue Divisor. Der alte Divisor wird zum Dividenden. Nun setzt man das Verfahren fort. Nach endlich vielen Schritten erhält man den ggT. Da das kleinste gemeinsame Vielfache (kgV) zweier Zahlen der Quotient aus ihrem Produkt und ihrem ggT ist, kann man auf der Basis des ggT sehr einfach das kgV berechnen.

**Beispiele:**

ggT von 544 und 391?

544	:	391	= 1;	Rest 153
391	:	153	= 2;	Rest 85
153	:	85	= 1;	Rest 68
85	:	68	= 1;	Rest 17
68	:	17	= 4;	Rest 0

der ggT von 544 und 391 ist 17

Daraus folgt:

Das kgV von 544 und 391 ist  
(544 · 391) : 17 = 12512.

ggT von 13 und 7 ?

13	:	7	= 1;	Rest 6
7	:	6	= 1;	Rest 1
6	:	1	= 6;	Rest 0

der ggT von 13 und 7 ist 1

Daraus folgt:

Das kgV von 13 und 7 ist  
7 · 13 = 91.

**Hinweise:** Kürzt man einen Bruch mit dem grössten gemeinsamen Teiler von Zähler und Nenner, entsteht ein Bruch, der nicht weiter kürzbar ist.

Wenn man zwei Brüche mit unterschiedlichem Nenner addieren möchte, muss man die Brüche zuerst gleichnamig machen. Das kleinste gemeinsame Vielfache ist der ideale Nenner dafür.

**Lösung:**

```
static void Main(string[] args) {
    int zahl1 = 0, zahl2 = 0, ggT = 0, kgV=0;
    do {
        Console.WriteLine("Zahl 1 eingeben:");
    } while (int.TryParse(Console.ReadLine(), out zahl1) == false);
    zahl1 = Math.Abs(zahl1); //Vorzeichen weglassen (Absolutwert)
    do {
        Console.WriteLine("Zahl 2 eingeben:");
    } while (int.TryParse(Console.ReadLine(), out zahl2) == false);
    zahl2 = Math.Abs(zahl2); //Vorzeichen weglassen (Absolutwert)
    int z1 = zahl1, z2 = zahl2, rest = 0;
    while (true) {
        rest = z1 % z2;
        if (rest == 0)
            break;
        z1 = z2;
        z2 = rest;
    }
    ggT = z2;
    kgV = (zahl1 * zahl2) / ggT;
    Console.WriteLine("Der ggT der beiden Zahlen {0} und {1} ist: {2}", zahl1, zahl2, ggT);
    Console.WriteLine("Das kgV der beiden Zahlen {0} und {1} ist: {2}", zahl1, zahl2, kgV);
}
```

## Aufgabe 15 (Würfelspiel)

**Thema:** Eingabe, Verarbeitung (mit Verzweigungen und Schleifen), Ausgabe

**Lernziele:**

- Die Lernenden können gegebene Aufgabenstellungen mit Einbezug von kopf- und fuss-gesteuerten Schleifen (inkl. Zählschleife) in c#- oder Java-Programmcode realisieren

**Zeit:** 180'



**Aufgabe:** Programmieren sie ein Würfelspiel. Die spielende Person muss zuerst den Namen und die Zielpunktzahl eingeben. Ist die Zielpunktzahl z.B. 100, gewinnt die Person, welche zuerst 100 erreicht hat.

Das Programm entscheidet zufällig, welche/r Spieler/-in beginnen darf.

Spieler\*in 1 beginnt zu würfeln. Dies wird mit zufallszahlen von 1 bis und mit 6 simuliert.

Nach jedem Würfeln muss die würfelnde Person entscheiden, ob sie weiterwürfeln um noch mehr Punkte holen möchte oder ob die in dieser Würfelsequenz erspielten Punkte gesichert werden sollen. Achtung!! Wenn man eine 1 würfelt, verliert man alle in der aktuellen Würfelsequenz erspielten Punkte. Nur die bereits gesicherten Punkte gehen nicht verloren, da sie in Sicherheit sind. Wenn man sich für das Sichern der Punkte entscheidet kommt der/die Gegner\*in an die Reihe.

Erstellen sie das Spiel so, dass sie gegen den Computer spielen können.

**Beispiel:** Jakobus spielt gegen Kunigunde. Sie spielen auf 30. Kunigunde darf beginnen. Sie würfelt eine 3, dann eine 2, eine 4, eine 5 und entscheidet sich danach zur Sicherung der Punkte. Ihr werden 14 Punkte gutgeschrieben. Nun ist Jakobus an der Reihe. Er würfelt eine 6, eine 4 und danach eine 1. Er verliert damit die Punkte der letzten Würfelsequenz (in diesem Fall alle Punkte). Sein Kontostand ist 0 Punkte. Nun ist Kunigunde an der Reihe. Sie würfelt eine 5, eine 3 und danach eine 1. Sie verliert damit die Punkte der letzten Würfelsequenz. Ihr Kontostand bleibt bei 14 und Jakobus ist an der Reihe, etc..... Wer zuerst über den Wert 30 kommt hat gewonnen!

**Ihre Lösung (kopieren Sie am Schluss hier Ihre Lösung hinein):**

```
static void Main(string[] args)
{

}

}
```