

Rekursionen

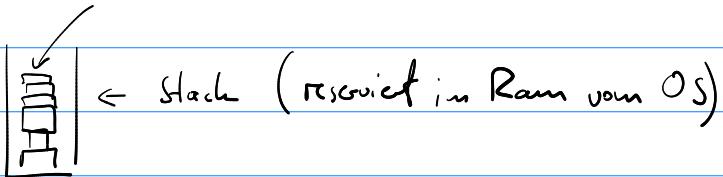
```
void x ()  
{  
    x ();  
}
```

Rekursion ist eine Methode,
die sich selbst aufruft!

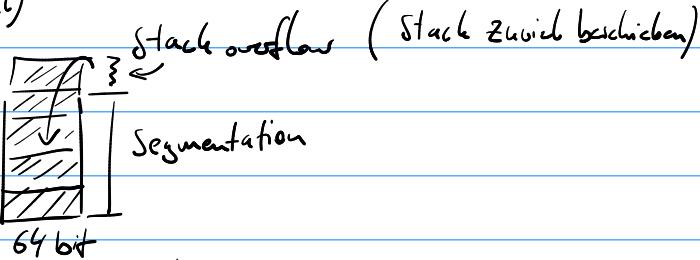
main ...

```
{  
    LIFO : Last in / first out
```

```
    x ();  
}
```



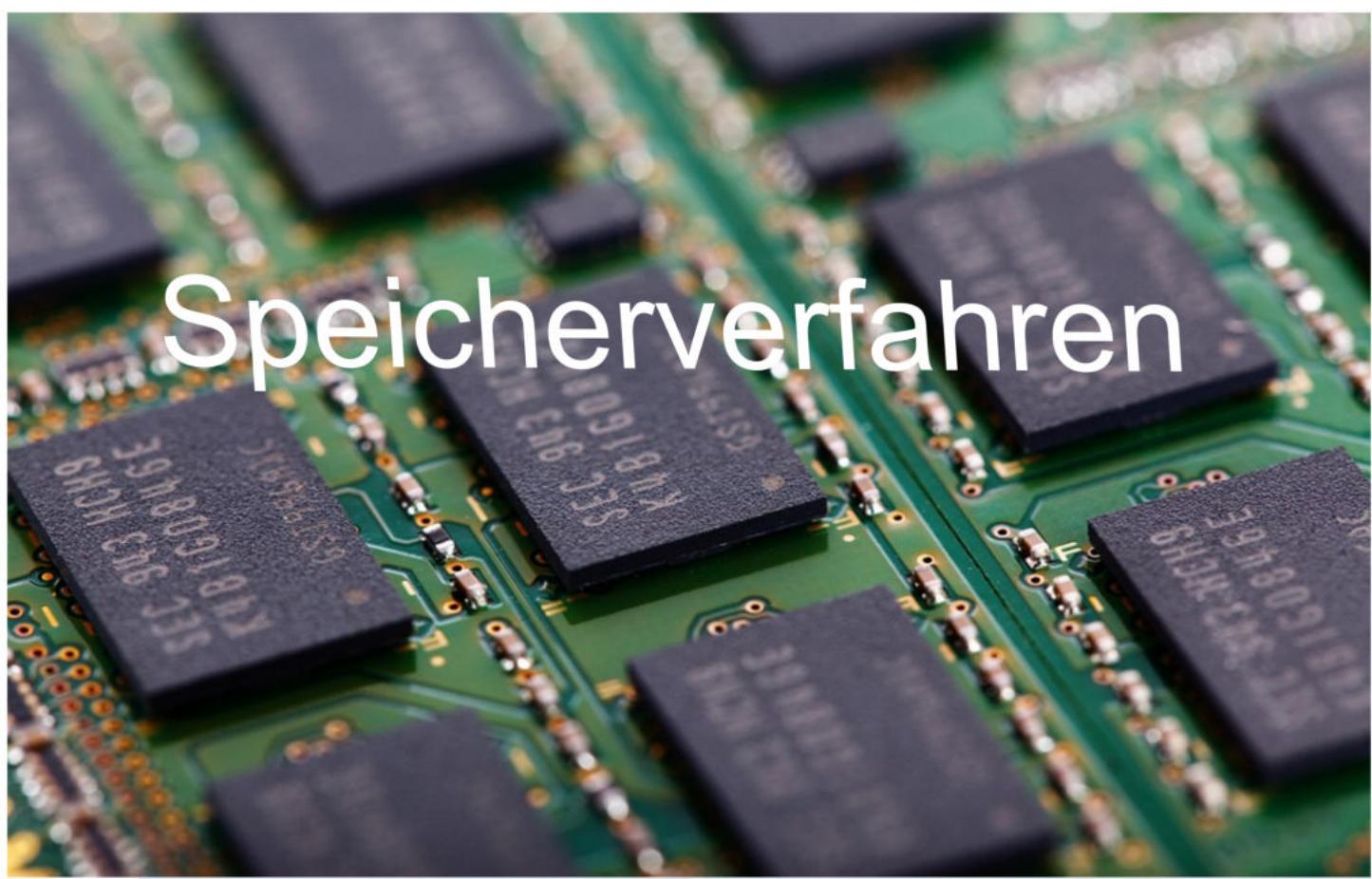
```
void x () ← Sprung (Call)  
{  
    x ();  
}
```



main ...

```
{  
    x ();  
}
```

Speicherverfahren

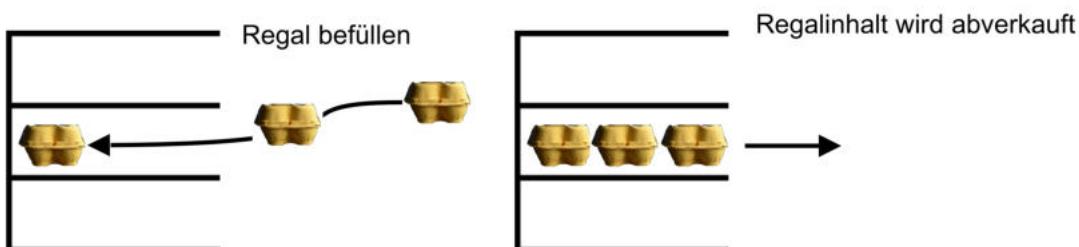


Einleitung

Verderbliche Waren, z.B. Hühnereier, sollen in einem Supermarkt verkauft werden. Aus Platzgründen müssen die Eierkartons ein einem Regal verstaut werden.

Erkennen Sie das Problem? => Stack

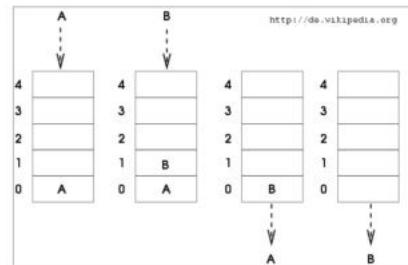
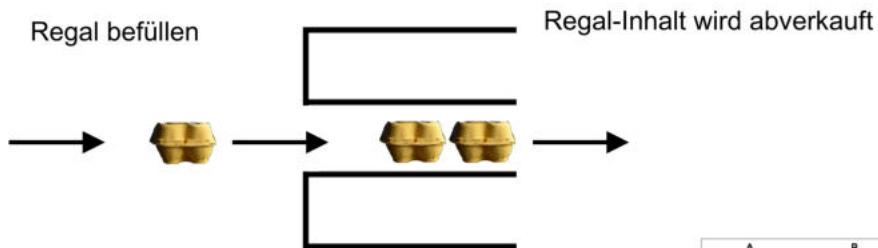
LIFO Last in, first out



Besser

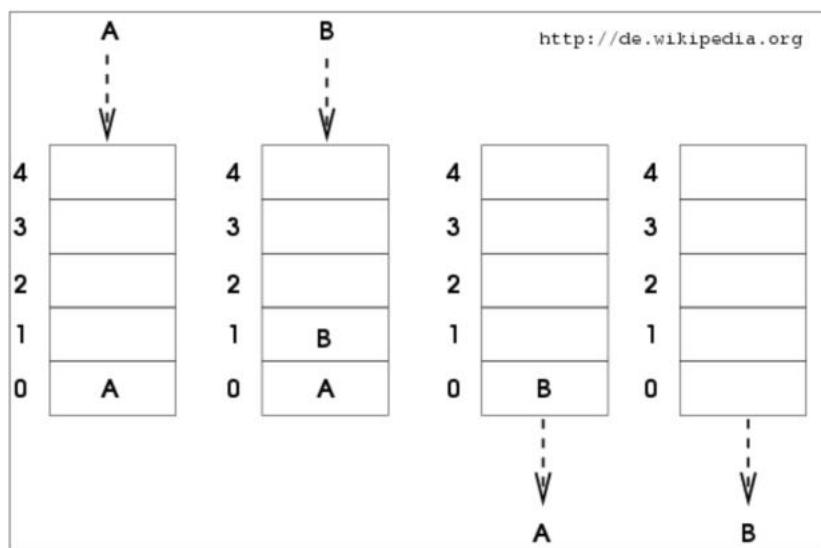
Warum ist das, im Falle von Lebensmitteln, besser?

Damit das erste befüllte LM vom Kunden genommen wird.



FiFo Prinzip

First in – First Out Prinzip (der Reihe nach)

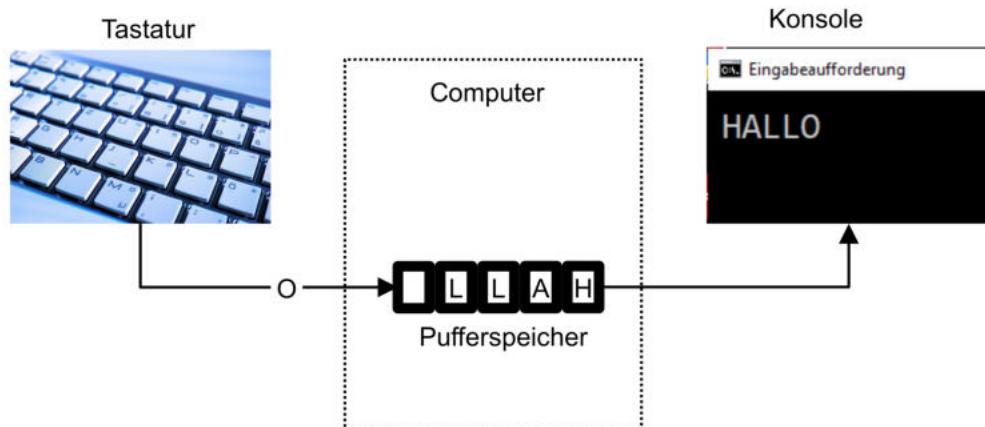


Von Fibi - <http://de.wikipedia.org/w/index.php?title=Datei:Fifo.png&filetimestamp=20040708055803&#file>, GFDL, <https://commons.wikimedia.org/w/index.php?curid=34774563>

FiFo Prinzip

First in – First Out Prinzip (der Reihe nach)

Beispiel Tastatur Puffer, es wird das Word „HALLO“ eingegeben und in der Konsole angezeigt.



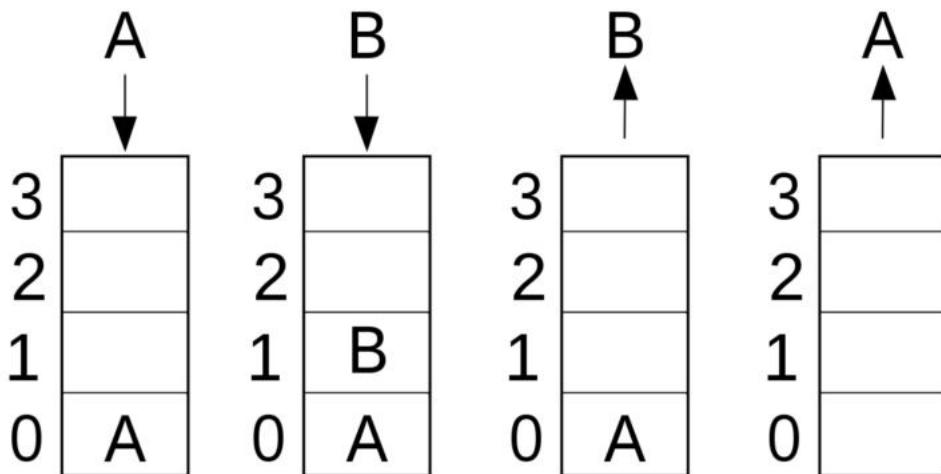
Stapelspeicher

Wie Funktioniert ein Stapel?



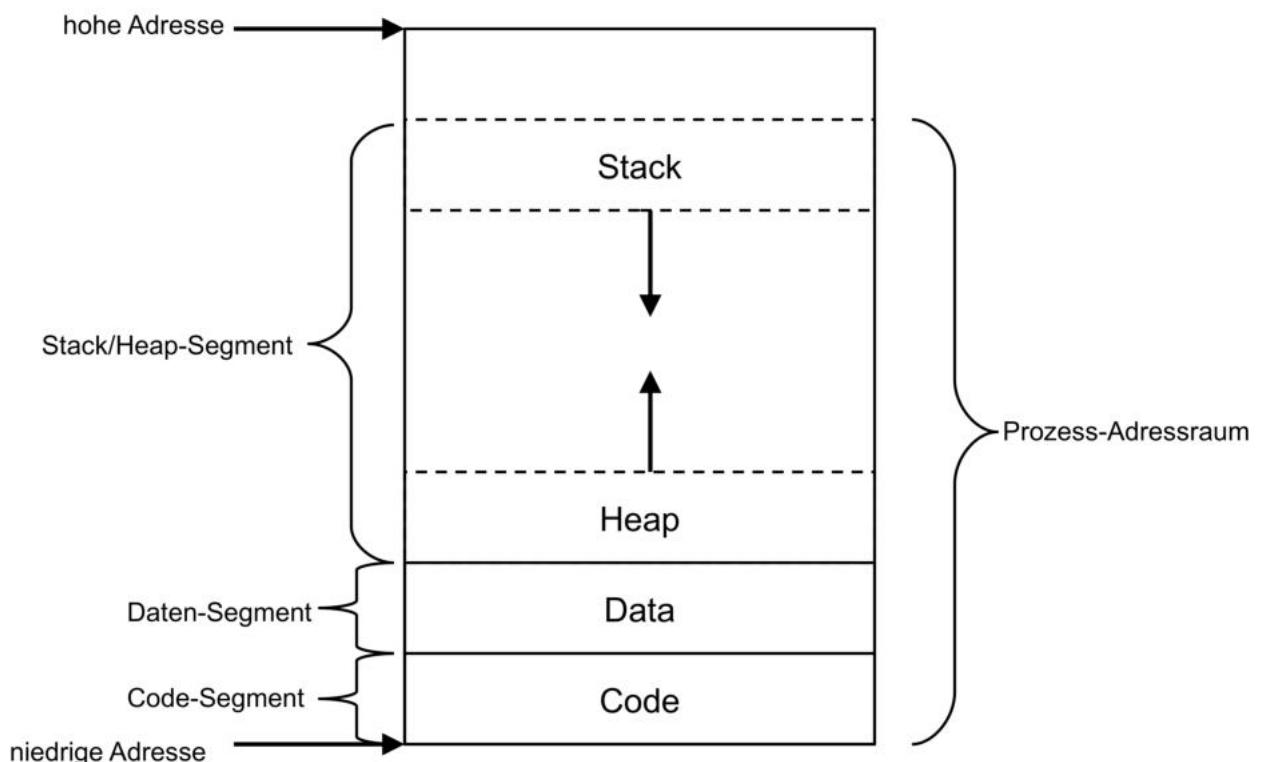
LiFo Prinzip

Last in – First out Prinzip (Stapel)



Von User:Gerrys - eigene Arbeit, CC BY-SA 3.0, <https://de.wikipedia.org/w/index.php?curid=4125115>

Speichersegmente in der Informatik (Prinzip)



Speichersegmente in der Informatik

- **Das "Code" Segment (Zugriff: wahlfrei)**

Das Codesegment hier wird der Programm-Code gespeichert und ausgeführt.

- **Das "Daten" Segment (Zugriff: wahlfrei)**

Hier werden statische Daten gespeichert. Dieses Segment enthält z.B. Konstanten bzw. Daten deren Wert/Inhalt zur Übersetzungszeit bereits fest stehen.

- **Das "Stack" Segment (Zugriff: Stapel, wächst von hoher zur niedrigen Adresse)**

Auf dem Stack werden lokale Variablen gespeichert, d.h. Variablen einer Funktion.

Auch die Parameter einer Funktion werden dort abgelegt.

- **Das "Heap" Segment (Zugriff: wahlfrei, wächst von niedriger zu hoher Adresse)**

Auf dem Heap werden Daten, von denen man nur während der Ausführung weiß wie groß sie sein werden, gespeichert. (Bsp.: _____)

Reflektion

- Wie ist ein Stack Organisiert.
- Welche Speichersegmente kennen Sie?
- Wozu dient das Heap-Segment
- Wozu das Code-Segment?

Methodenüberlagerung



Definition Methodenüberlagerung

- Wenn zwei oder mehr Methoden mit gleichem Namen jedoch unterschiedlichen Parametern existieren, dann spricht man von Methodenüberlagerung.

Beispiel:

Es soll eine Methode „AddInt(...)“ für die Addition von 2 Integer Zahlen oder 3 Integer Zahlen implementiert werden.

Formulieren Sie die Deklaration der zwei Methoden.

Aufruf Methodenüberlagerung

```
class Program
{
    public static int AddInt( int a, int b )
    {
        return a + b;
    }

    public static int AddInt( int a, int b, int c )
    {
        return a + b + c;
    }

    static void Main( string[] args )
    {
        Console.WriteLine(AddInt(223, 47));
        Console.WriteLine(AddInt(123, 4984, 3904));
    }
}
```

Welche Methode wird aufgerufen?

- Es wird während der Compilierung entschieden welche Methode aufgerufen werden soll.
- Der Compiler nimmt die Methode, deren Datentypen und Anzahl mit dem Aufruf übereinstimmt

⇒ Der Compiler verwendet die Signatur von Methoden und vergleicht diese mit den Methoden Aufrufen.

Dort wo die Signatur mit dem Aufruf übereinstimmt, wird die entsprechende Methode verwendet.

Die Signatur besteht aus: Name, Parametertyp und Anzahl Rückgabewert (T_{RP})

Welcher Aufruf gehört zu welcher Methode?

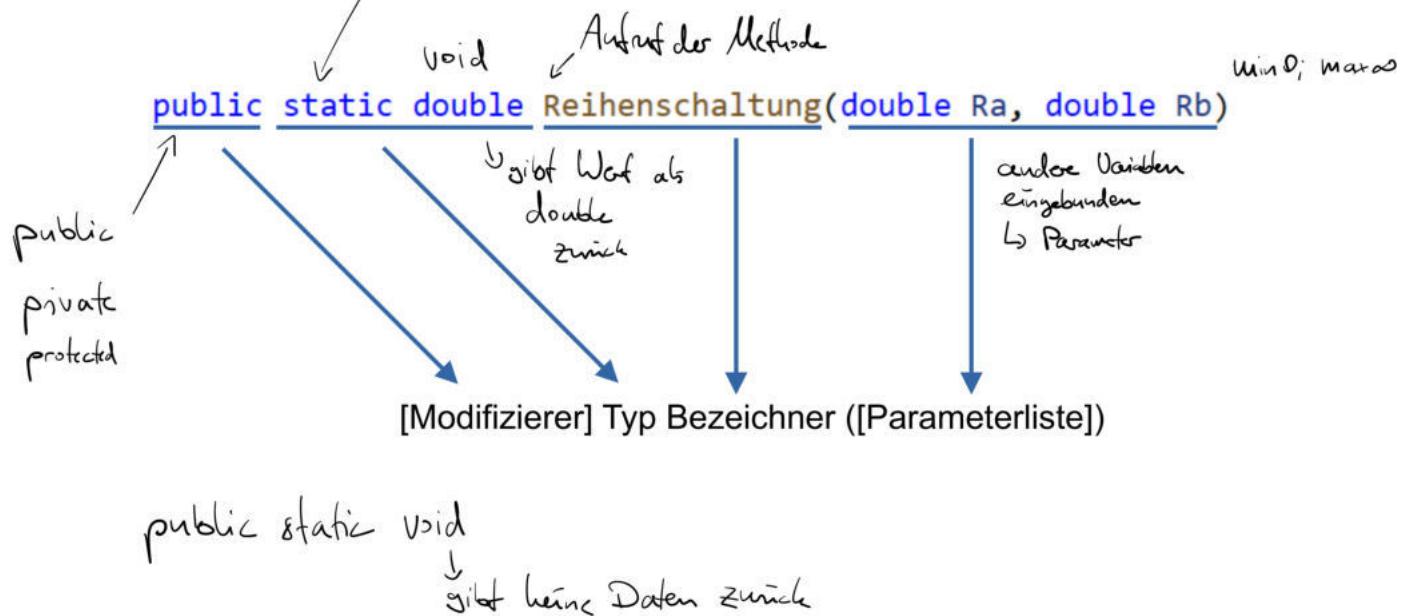
```
1 public static int AddNumbers(int a, int b)
{
    return a + b;
}
2 public static int AddNumbers(int a, double b)
{
    return a + (int) b;
}
3 public static int AddNumbers(int a, int b, int c)
{
    return a + b + c;
}
static void Main(string[] args)
{
    Console.WriteLine(AddNumbers(123, 4984, 3904));
    Console.WriteLine(AddNumbers(123, 39.3));
    Console.WriteLine(AddNumbers(223, 47));
}
```

Methoden



immer vorhanden im Speicher

Aufbau



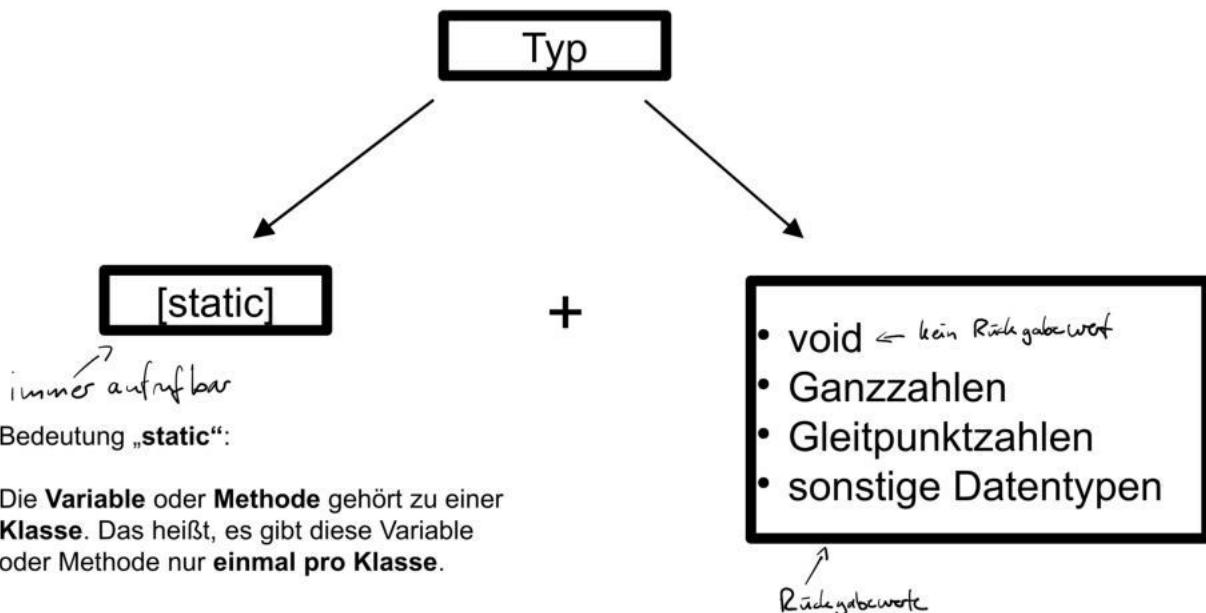
[....] : Optional, kann weg gelassen werden. ACHTUNG: Weglassen hat Konsequenzen.

Modifizierer

Modifizierer	Beschreibung
public	Der Zugriff unterliegt keinen Einschränkungen.
private	Zugriff nur innerhalb einer Klasse möglich.
protected	Zugriff innerhalb einer Klasse und in abgeleiteten Klassen möglich.
internal	Auf den Typ oder Member kann nur in der gleichen Assembly zugegriffen werden.
protected internal	Kombination aus protected und internal.
private internal	Kombination aus private und internal.

Typ

Der Typ einer Methode setzt sich zusammen aus:



Ohne Rückgabewert/ Ohne Parameter

```
3 public static void Hallo1()
{
    Console.WriteLine("Hallo");
}
1 Verweis
4 public static void Hallo2()
{
    int a,b;
    a = 5000;
    b = 6000;
    Console.WriteLine("Hallo");
    return;
}
}           ↪ bringt Methode unmittelbar/sofort ab
```

Korrektur? public static void Hallo2(int a, int b)

return → void

Ausgabe: Hallo
Haloo
a ist 123
b ist 456

0 Verweise

```
static void Main(string[] args)
{
    1 int a = 123;
    2 int b = 456;
    3 Hallo1();
    4 Hallo2();
    5 Console.WriteLine("a ist " + a);
    6 Console.WriteLine("b ist " + b);
}
```

mit Rückgabewert/ Ohne Parameter

```
public static int methode1()
{
    return 444; → gibt den integer Wert '444' zurück
}                                            Beendet Methode sofort
                                                Korrektur?
1 Verweis
public static int methode2()
{
    int a,b;
    a = 1;
    b = 2;
    return 555;
}
0 Verweise
static void Main(string[] args)
{
    int a = 123;
    int b = 456;

    a = methode1();
    Console.WriteLine(a);
    a = methode2();
    Console.WriteLine(a);
    Console.WriteLine(b);
}
```

Return → 444
555

Ausgabe: 444
555
456

6

Ohne Rückgabewert/ mit Parametern

Call by Value

```
public static void cbv1(int x, int y)
{
    int x,y;
    Console.WriteLine(x);
    Console.WriteLine(y);
    x = 1;
    y = 2;
}
1 Verweis
public static void cbv2(int a, int b)
{
    a = 4711;
    b = 4712;
    return;
}
0 Verweise
static void Main(string[] args)
{
    int a = 123;
    int b = 456;
    cbv1(a, b);
    cbv2(a, b);
    Console.WriteLine(a);
    Console.WriteLine(b);
}
```

Return → 123
456

Ausgabe: 123
456
123
456

Call by Value bedeutet, dass Parameter an Funktionen mit Hilfe einer Kopie übergeben werden.

Das bedeutet, dass innerhalb der aufgerufenen Funktion mit der Kopie gearbeitet wird und nicht mit den Originalen!

Änderungen wirken sich nicht auf die ursprünglichen Werte aus.

7

mit Rückgabewert/ mit Parametern

Call by Value

```
public static double cbv3(int x, int y)
{
    y = x;
    return 3.141*x/y;
}
0 Verweise
static void Main(string[] args)
{
    int a = 123;
    int b = 456;
    double c;
    Console.WriteLine(cbv3(a,b));
    c = cbv3(a, b);
    Console.WriteLine(cbv3(a, b));
}
```

Return →

Ausgabe:

Call by Reference

Call by Reference

```
public static void exchange(ref int erster, ref int zweiter)
{
    int puffer;
    puffer = erster;
    erster = zweiter;
    zweiter = puffer;
}
0 Verweise
static void Main(string[] args)
{
    int a = 123;
    int b = 456;
    Console.WriteLine("a = " + a + "   b = " + b);
    exchange(ref a, ref b);
    Console.WriteLine("a = " + a + "   b = " + b);
    Console.ReadKey();
}
```

Ausgabe: a = 123 b = 456
a = 456 b = 123

Call by Reference bedeutet, dass Parameter an Funktionen mit Hilfe eines Verweises auf dessen Speicheradresse übergeben werden.

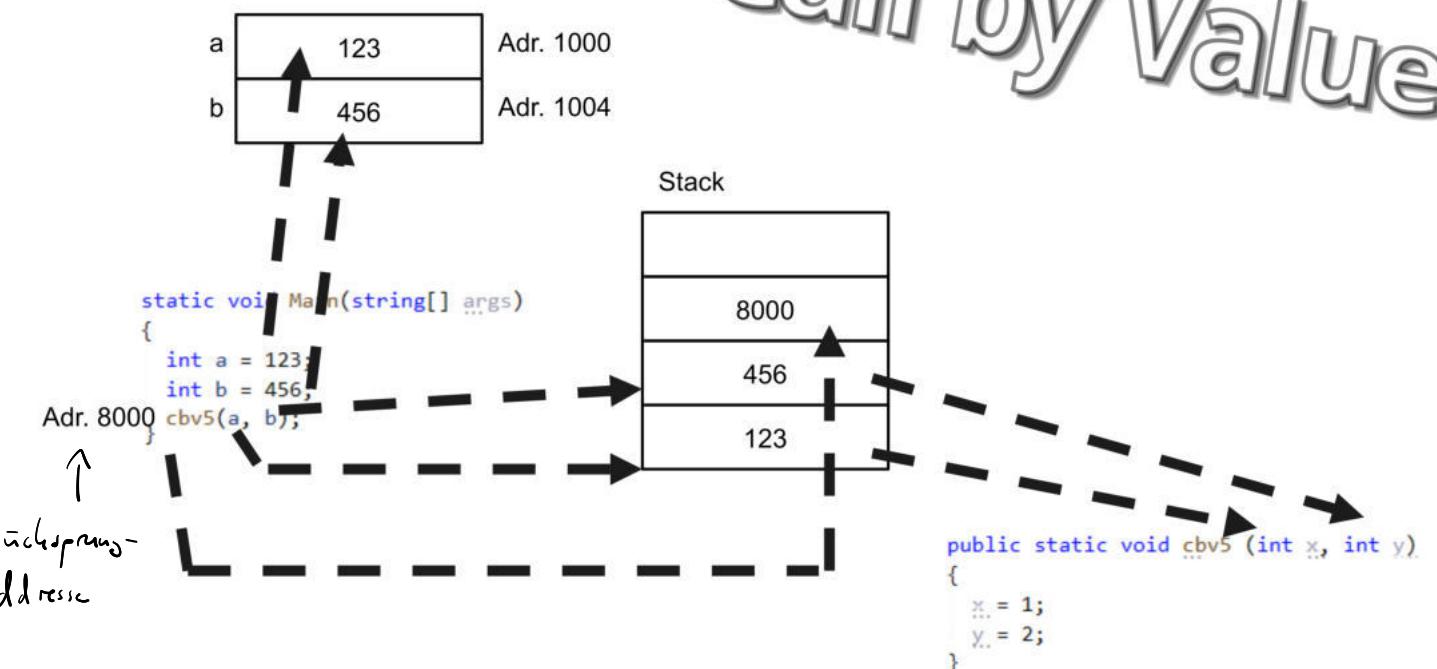
Das bedeutet, dass innerhalb der aufgerufenen Funktion mit dem Original gearbeitet wird!

Änderungen wirken sich auf die ursprünglichen Werte aus.

Zugreifen auf Original

„Call by Value“ vs. „Call by Reference“

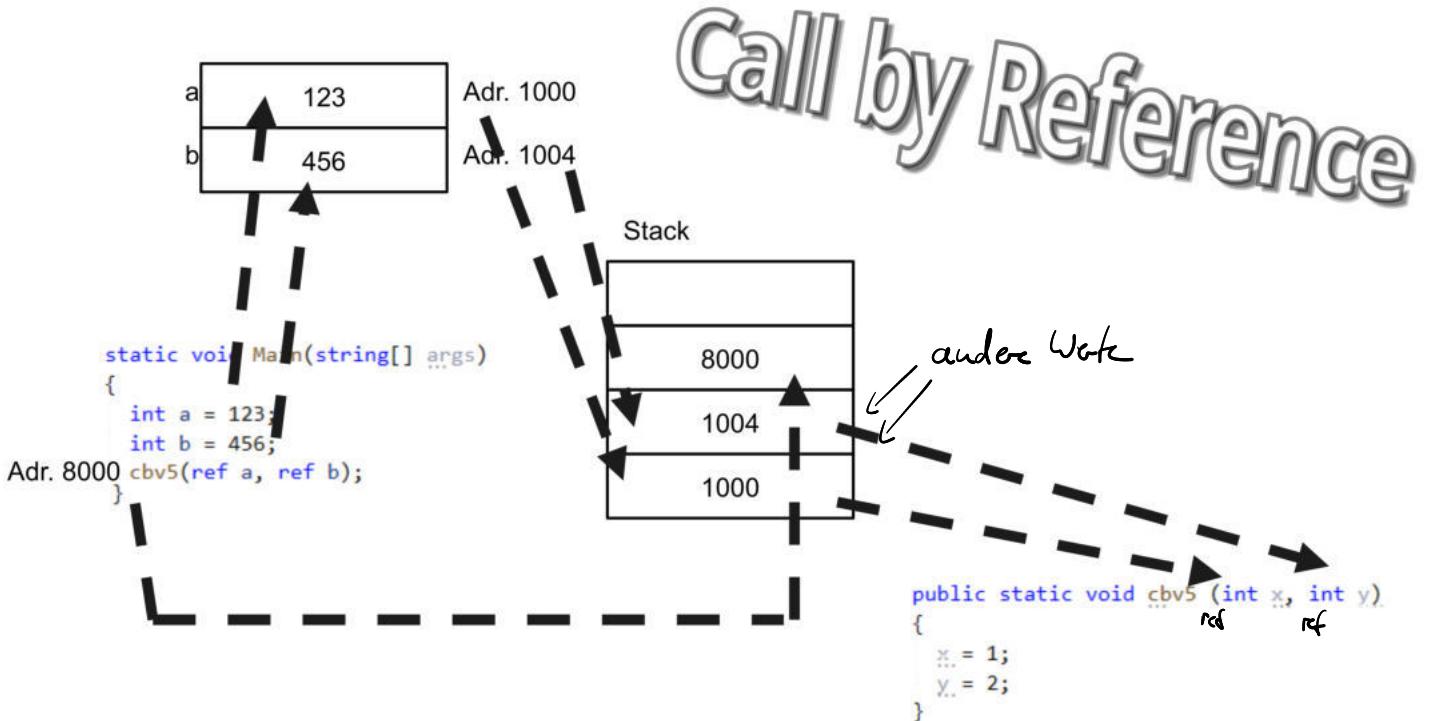
Call by Value



„Call by Value“ vs. „Call by Reference“

Call by Value

„Call by Value“ vs. „Call by Reference“



„Call by Value“ vs. „Call by Reference“

Call by Reference

Reflektion

- Können Call by Value Variablen in einer aufgerufenen Methoden verändert werden, so dass der Aufrufer diese Änderung sieht?

Nein, da es in der Methode bleibt.

- Können Call by Reference Variablen in einer aufgerufenen Methoden verändert werden, so dass der Aufrufer diese Änderung sieht?

Ja klar.

- Können Call-by-Reference Methoden einen Rückgabewert haben der mit Return zurück gegeben wird?

Ja, um zur Fehlersignierung

- Können in einer Methode Reference und Value Parameter gemischt werden?

Ja

Der einarmige Bandit (Slot Machine) Call by Reference



Auftrag

- Sie arbeiten in einer Firma, die einen Glücksspielautomat entwickelt.



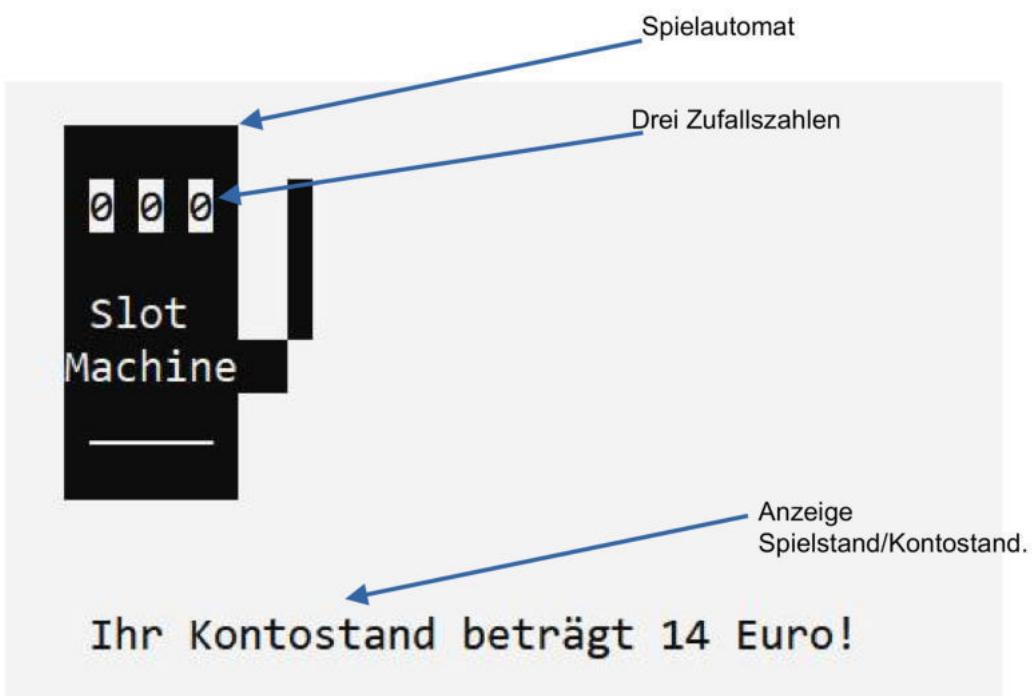
- Leider hat der Programmierer von Heute auf Morgen gekündigt, das Programm ist nicht lauffähig.



- Nur der Quellcode des Programmierers steht Ihnen zur Verfügung.

- Auftrag Ihrer Chefin: Analysieren und Ergänzen Sie den Quellcode. **Wir brauchen ein Produkt, dass wir verkaufen können!**

Ziel



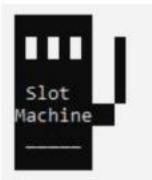
Vorgehen

- Kopieren Sie das „Einarmiger_Bandit_Vorlage.zip“ in Ihr lokales Verzeichnis.
- Entpacken Sie die Zip Datei, öffnen Sie das Projekt.
- Analysieren Sie mit dem Debugger folgende Elemente:

zeichneSpielfeld()

AnzeigeZahlenwerte(int Wert1, int Wert2, int Wert3)

- Erklären Sie um was es sich bei diesen zwei „Konstrukten“ handelt.



Wie funktioniert ... eine Methode / Funktion

public static void zeichneSpielfeld()

[Modifizier]

L Typ

L Bezeichner

[] Optional, kann jetzt weglassen werden!

static → Methode ist immer vorhanden (statisch)

void → kein Rückgabewert; kein Datentyp



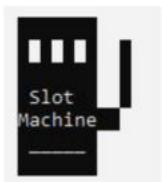
Wie funktioniert ...

public static void zeichneSpielfeld()

Eine Methode wird in C# von anderen Methoden aufgerufen
Ausnahme die Main Methode. Diese startet zuerst.

Bsp: Pseudocode

```
... Main(...)  
{ zeichneSpielfeld(); } auf ruf der Methode
```



Wie funktioniert ...

public static void AnzeigeZahlenwerte(int Wert1, int Wert2, int Wert3)

↓ [Modifizierer] → Typ ↓ Bezeichner ↓ [Parameterliste]

[Optional]

Anzahl Parameter von 0 bis ∞

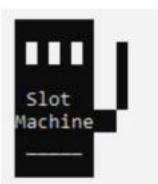
Das hier ist eine Call by Value Methode!



Wie funktioniert ...

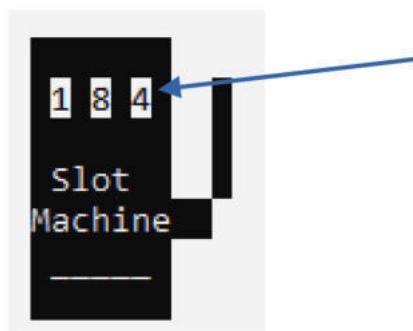
```
public static void AnzeigeZahlenwerte(int Wert1, int Wert2, int Wert3)
```

Call by Value bedeutet, dass die übergebenen Parameter nur als "Wert", aber nicht als „Variable“ übergeben werden!



Aufgabe 1 / Initialisierung

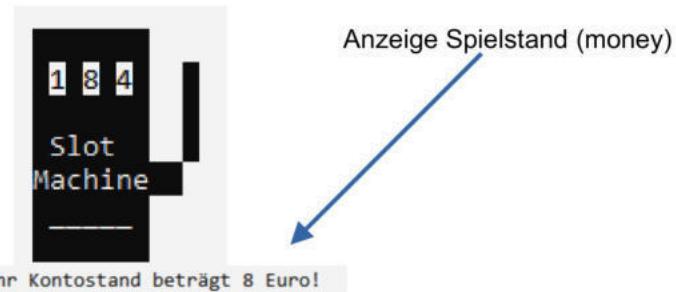
- Drei Zufallszahlen sollen in der Anzeige angezeigt werden.
- Verwenden Sie dazu „AnzeigeZahlenwerte(...)“



Anzeige drei Zufallszahlen

Aufgabe 2 / Anzeige

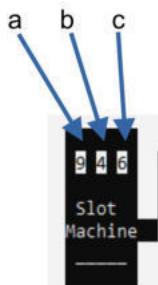
- Lassen Sie die Zufallszahlen in einer While Schleife erzeugen, so dass in jedem Durchlauf ein neues 3er Paar Zufallszahlen im Spielautomat angezeigt wird.
- Sobald eine Zufallszahl angezeigt wird, soll auf die Eingabe von Return/Enter gewartet werden.
- Lassen Sie den Spielstand (money) anzeigen. (Spielstand soll am Anfang 10 Euro betragen.)
- Die Spielstand Anzeige soll über eine Methode „Spielstand(....)“ erfolgen.



Aufgabe 3 / Logic

- Entwickeln Sie eine Spiellogik die den Spielstand/ Kontostand in jedem Durchlauf beeinflusst.
- Vorschlag:

Wenn a = b dann money = money + 2€.
Wenn a = c dann money = money + 2€.
Wenn b = c dann money = money + 2€.



Ihr Kontostand beträgt 0 Euro!
Das Spiel ist vorbei, sie haben 29 runden gespielt!

Preis pro Spiel (Durchlauf) 1€.

Wenn money = 0 dann Spielende.

Lassen Sie die Spielrunden zählen.

Lassen Sie die Anzahl der gespielten Runden ausgeben
wenn das Spiel zu Ende ist.

Aufgabe 4 / Erweiterungen

- Erweitern Sie das Spiel durch Klänge.



Ihr Kontostand beträgt 0 Euro!

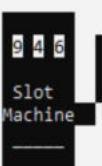
Das Spiel ist vorbei, sie haben 29 runden gespielt!

- Suchen Sie, z.B. über Google, wie man einen Rückgabewert einer Methode implementiert.
- Lagern Sie die Spielelogik in eine Methode „Logik(...)“ aus. Die Methode benötigt dazu einen Integer-Wert als Rückgabewert. Dieser wird auf den Spielstand addiert.

money=money + Logik(wert1, wert2, wert3);

Reflektion

- Führen Sie Ihr Programm vor.



Ihr Kontostand beträgt 0 Euro!

Das Spiel ist vorbei, sie haben 29 runden gespielt!

Lottozahlen sortieren

Fabian Steiß

für 'i' = 0 von 'i' bis Länge 'LottoZahlenSortiert' zähle 'i' hoch

min = 50

für 't' = 0 von 't' bis Länge 'LottoZahlen' zähle 't' hoch

ja

Wenn LottoZahlen[t] < min

nein

min = LottoZahlen[t];

minpos = t;



LottoZahlenSortiert[i] = min;

LottoZahlen[minpos] = 50;

Schreibe "Die Lottozahlen sortiert:"

für 'i' = 0 von 'i' bis Länge 'LottoZahlenSortiert' zähle 'i' hoch

Schreibe: "LottoZahlenSortiert[i] + " "

Schreibe Linie

Initialisierung 3D Array

- 2. Dimension

```
int [,,] Array3D = {  
    {{1,2,3,4},{3,4,5,6},{6,7,8,9}},  
    {{9,2,4,1},{3,19,5,0},{4,5,8,9}}  
};
```

Array3D[2, 3, 4]

2

22.01.21 11:12:26

Initialisierung 3D Array

- 1. Dimension

```
int [,,] Array3D = {  
    {{1,2,3,4},{3,4,5,6},{6,7,8,9}},  
    {{9,2,4,1},{3,19,5,0},{4,5,8,9}}  
};
```

Array3D[2, 3, 4]

3

22.01.21 11:12:26

Initialisierung 3D Array

- Console.WriteLine(Array3D[1,1,1])
- Welche Zahl wird ausgegeben?

```
int [,,] Array3D = {  
    {{1,2,3,4},{3,4,5,6},{6,7,8,9}},  
    {{9,2,4,1},{3,19,5,0},{4,5,8,9}}  
};
```

4

22.01.21 11:12:26

Initialisierung 3D Array

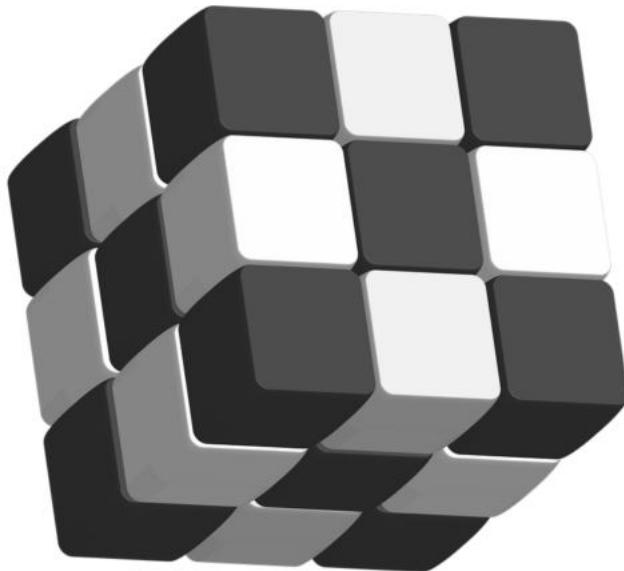
- Console.WriteLine(Array3D[1,1,1])
- Welche Zahl wird ausgegeben?

```
int [,,] Array3D = {  
    {{1,2,3,4},{3,4,5,6},{6,7,8,9}},  
    {{9,2,4,1},{3,19,5,0},{4,5,8,9}}  
};
```

5

22.01.21 11:12:27

Mehrdimensionale Arrays



Definition Allgemein zu Arrays

Ein Array ist eine **Datenstruktur**, die eine Anzahl von Variablen enthält, auf die über berechnete **Indizes** zugegriffen wird.

Die im Array enthaltenen Variablen, auch Elemente des Arrays genannt, weisen **alle denselben Typ auf**. Dieser Typ wird als **Elementtyp** des Arrays bezeichnet.

Die Deklaration einer Arrayvariablen reserviert Speicher für einen Verweis auf eine Arrayinstanz. → Speicherbereich, der für Array reserviert wird

Die tatsächlichen Arrayinstanzen werden unter Verwendung des **new-Operators** zur Laufzeit **dynamisch erstellt**.

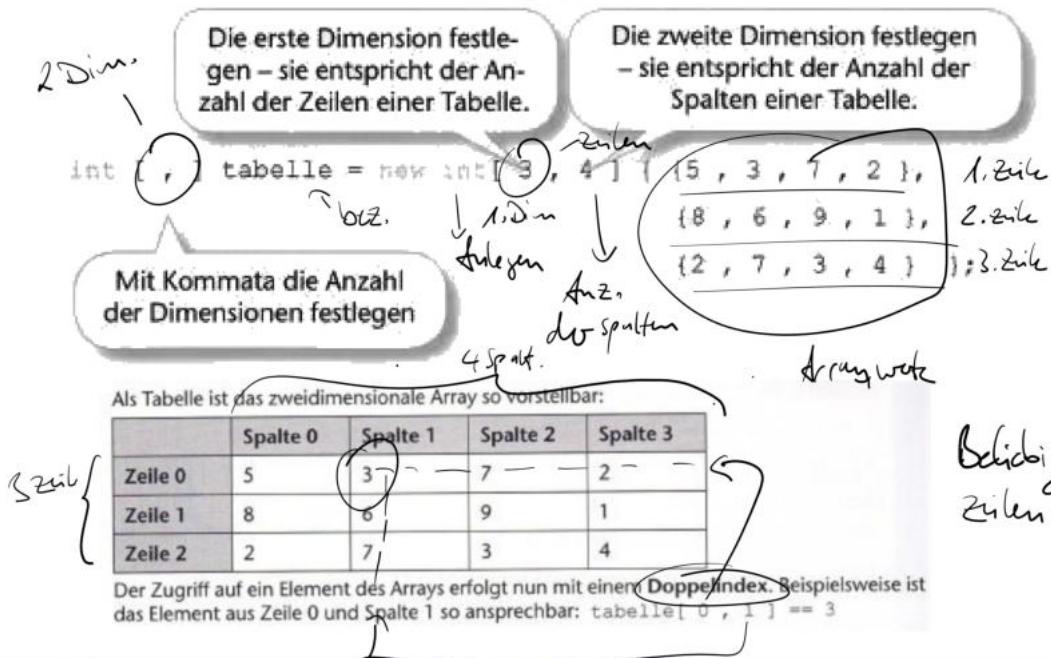
↳ in dem Moment, wie ich

Der new-Vorgang legt die **Länge der neuen Arrayinstanz** fest, die dann **für die Lebensdauer** der Instanz beibehalten wird.

Die Indizes der Arrayelemente reichen von 0 bis Length - 1.

Syntax mehrdimensionales Array

Es wird ein zweidimensionales Array angelegt.

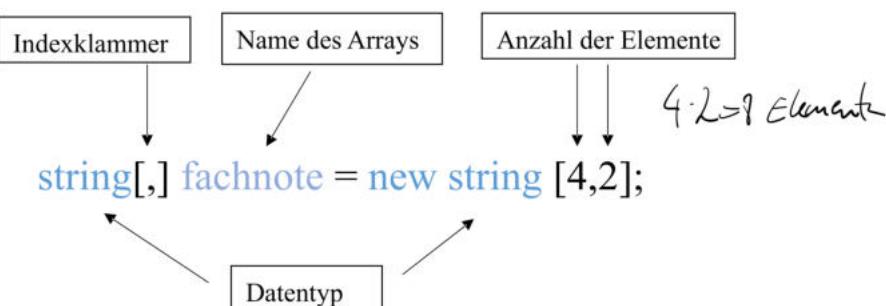


3

2-Dimensionales Array

Die Syntax ein 2-dimensionalen Array deklariert.

1 Klammer = 2 Dim.



4

2-Dimensionales Array, Beispiel

Es sind die Noten eines Schülers zu erfassen. Diese sollen geordnet zwischengespeichert und ausgegeben werden.

Der Schüler XY hat die Noten: Mathe = 1,5 , Informatik = 2,5 , Elektrotechnik = 2,0 , Bkom = 1,7

Anhand einer Tabelle ordnen wir die Wertepaare an, in C# hier anhand eines 2-dimensionalen Arrays, im Allgemeinen anhand eines mehrdimensionalen Arrays.

```
class Program
{
    static void Main(string[] args)
    {
        string[,] fachnote = new string[ 4, 2];
        fachnote[0, 0] = "Mathe";           fachnote[0, 1] = "1,5";
        fachnote[1, 0] = "Informatik";     fachnote[1, 1] = "2,5";
        fachnote[2, 0] = "Elektrotechnik"; fachnote[2, 1] = "2,0";
        fachnote[3, 0] = "Bkom";          fachnote[3, 1] = "1,7";
        //for Schleife zum Auslesen des Arrays
        for (int i = 0; i < fachnote.GetLength(0); i++)
        {
            for (int j = 0; j < fachnote.GetLength(1); j++)
            {
                Console.WriteLine("Element ({0},{1})={2}", i, j, fachnote[i,j]);
            }
        }
        Console.ReadKey();
    }
}
```

0	1
0	Mathe 1,5
1	Infor 2,5
2	El. 2,0
3	Bkom 1,7

```
*****
Element (0,0)=Mathe
Element (0,1)=1,5
*****
Element (1,0)=Informatik
Element (1,1)=2,5
*****
Element (2,0)=Elektrotechnik
Element (2,1)=2,0
*****
Element (3,0)=Bkom
Element (3,1)=1,7
*****
```

3-Dimensionales Array

Beispiel:

Es wird ein dreidimensionales Array angelegt.

```
float [ , , ] tabellen = new float [ 3 , 3 , 4 ];
```

Das dreidimensionale Array ist dann so vorstellbar:

Blatt 2		Spalte 0	Spalte 1	Spalte 2	Spalte 3
Zeile 0	1 5		7	12 33	25 3
Blatt 1	Spalte 0	Spalte 1	Spalte 2	Spalte 3	
Zeile 0	1 5	7	12 33	25 3	2
Blatt 0	Spalte 0	Spalte 1	Spalte 2	Spalte 3	
Zeile 0	1.5	7	12.33	25.3	55
Zeile 1	124	99.99	453	67.89	6
Zeile 2	12	90.2	2727.5	22	

tabellen [1 , 1 , 3] == 45.55

Tabellenblatt

Zeile

Spalte

X-Dimensionale Arrays, Wissenswertes

- Der new-Operator initialisiert die Elemente eines Arrays automatisch mit ihren Standardwerten. Dieser lautet z.B. für alle numerischen Typen 0 und für alle Verweistypen null.
- Der new-Operator erlaubt es, die Anfangswerte der Arrayelemente unter Verwendung eines **Arrayinitialisierers** anzugeben, bei dem es sich um eine Liste von Ausdrücken zwischen den Trennzeichen { und } handelt. Mit dem folgenden Beispiel wird ein int[] mit drei Elementen zugewiesen und initialisiert.

```
int[] a = new int[] {1, 2, 3};
```

Die Länge des Arrays wird von der Anzahl an Ausdrücken zwischen { und } abgeleitet.

Deklarationen lokaler Variablen und Felder können weiter verkürzt werden, sodass der Arraytyp nicht erneut aufgeführt werden muss.

- int[] a = {1, 2, 3};

Die zwei vorherigen Beispiele entsprechen dem folgenden Code:

```
int[] t = new int[3];
t[0] = 1;
t[1] = 2;
t[2] = 3;
int[] a = t;
```

- Die Anzahl von Dimensionen eines Arraytyps ist 1 plus die Anzahl von Kommas, die innerhalb der eckigen Klammern des Arraytyps angegeben ist.

Arraytypen sind **Verweistypen** (s. Merkblatt zu „array (III)“ und „Wertetypen und Verweistypen“).

<Array-Bezeichner>.Length ruft die Gesamtanzahl der Elemente in allen Dimensionen im Array ab.

<Array-Bezeichner>.Rank ruft den Rang (Anzahl der Dimensionen) von Array ab.

<Array-Bezeichner>.GetLength(i) ruft eine 32-Bit-Ganzzahl ab, die die Anzahl der Elemente in der angegebenen Dimension des Array angibt.

<Array-Bezeichner>.GetUpperBound(i) wird verwendet, um den max. Index in jeder Dimension eines mehrdimensionalen Arrays zu bestimmen.

7

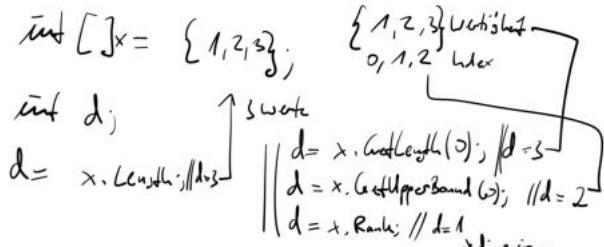
Beispiel:

```
using System;
public class Example
{
    public static void Main()
    {
        // Declare a single-dimensional string array
        String[] array1d = { "zero", "one", "two", "three" };
        ShowArrayInfo(array1d);

        // Declare a two-dimensional string array
        String[,] array2d = { { "zero", "0" }, { "one", "1" },
                             { "two", "2" }, { "three", "3" },
                             { "four", "4" }, { "five", "5" } };
        ShowArrayInfo(array2d);

        // Declare a three-dimensional integer array
        int[, , ] array3d = new int[, , ] { { { 1, 2, 3 }, { 4, 5, 6 } },
                                             { { 7, 8, 9 }, { 10, 11, 12 } } };
        ShowArrayInfo(array3d);
    }

    private static void ShowArrayInfo(Array arr)
    {
        Console.WriteLine("Length of Array: {0,3}", arr.Length);
        Console.WriteLine("Number of Dimensions: {0,3}", arr.Rank);
        // For multidimensional arrays, show number of elements in each dimension.
    }
}
```



Fortsetzung Beispiel

```
if (arr.Rank > 1) {
    for (int dimension = 1; dimension <= arr.Rank; dimension++)
        Console.WriteLine("    Dimension {0}: {1,3}", dimension,
                           arr.GetUpperBound(dimension - 1) + 1);
}
Console.WriteLine();
}

// The example displays the following output:
//      Length of Array:      4
//      Number of Dimensions: 1
//
//      Length of Array:      12
//      Number of Dimensions: 2
//          Dimension 1:     6
//          Dimension 2:     2
//
//      Length of Array:      12
//      Number of Dimensions: 3
//          Dimension 1:     2
//          Dimension 2:     2
//          Dimension 3:     3
```

Typische Fehler

Typische Fehler im Umgang mit (mehrdimensionalen) Array

Der häufigste Fehler beim Umgang mit Arrays ist es, die Grenzen des Arrays zu überschreiten und so auf undefinierte oder ungültige Speicherbereiche zuzugreifen.

Bsp: wir greifen auf Feldelement 10 zu, obwohl es nur bis 10 Elemente hat und daher nur bis 9 deklariert ist

```
Int Zahlenfeld[10]
Int i = Zahlenfeld [10]
→ falsch, da auf ein nicht deklariertes Feldelement zugegriffen wird
```

Ein weiterer typischer Fehler sind Fehler bei der Deklaration und Initialisierung.

Bsp: Wir erstellen ein 3x2 Array, Initialisieren in der Dimension 0 ein 4-tens Element

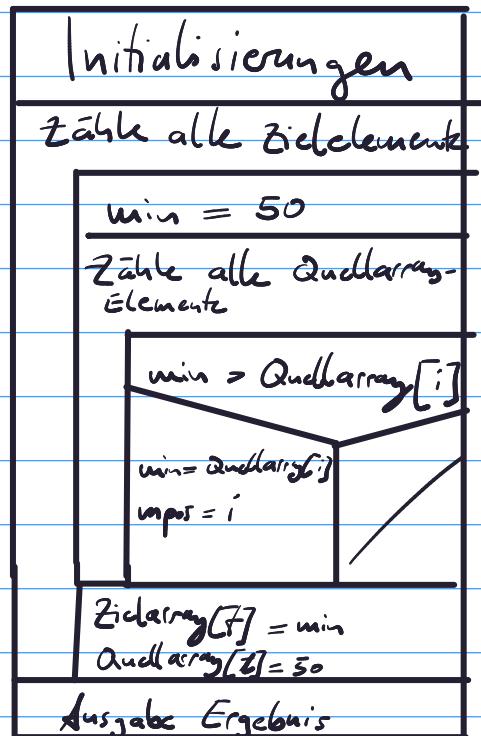
```
string[,] array2D = new string [3, 2] { {"eins", "zwei"}, {"drei", "vier"}, {"fünf", "sechs"}, {"sieben", "acht"} }
```

→ falsch, da auf ein vieres nicht initialisiertes Element in der ersten Dimension zugegriffen wird

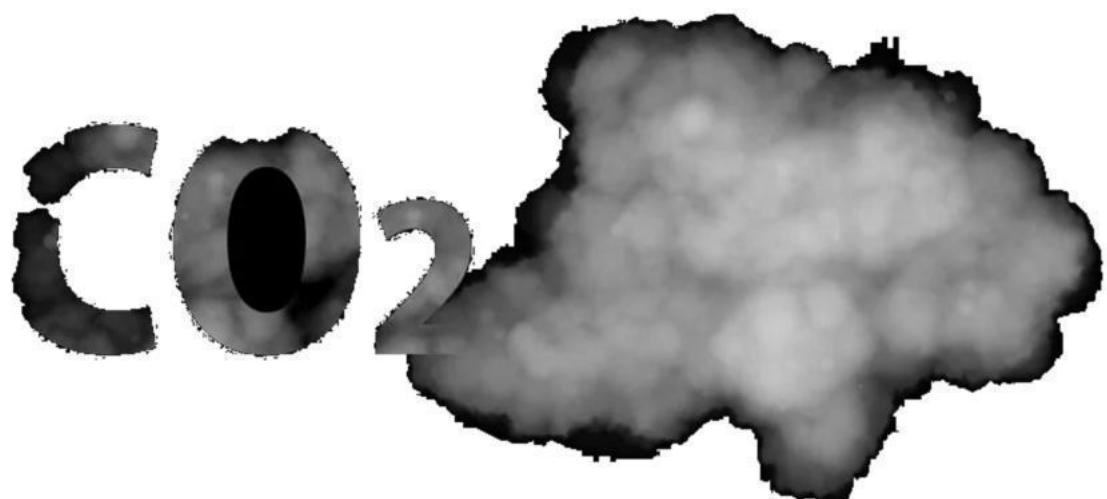
Naive Sortierungsverfahren

- Insertion Sort

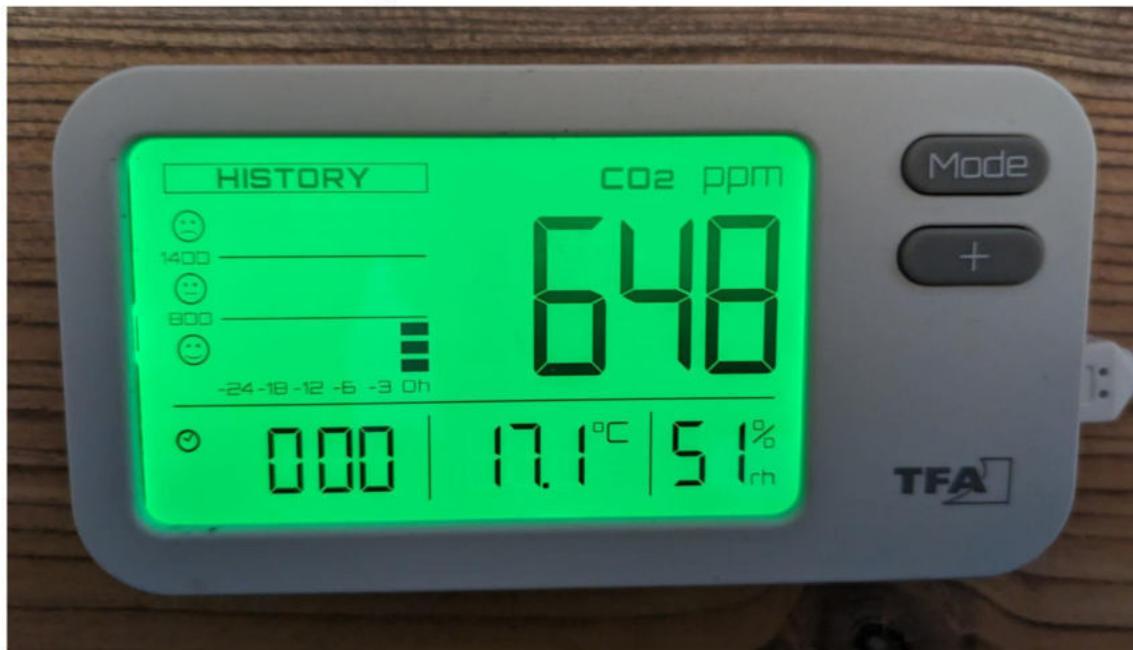
Struktogramm



CO₂ Messwerte im Klassenzimmer



CO2 Messgerät



Messung seit 40 Jahren...

- Das CO2 Messgerät misst seit 40 Jahren den CO2 Gehalt im Klassenzimmer.
- Es zeichnet alle 5 Minuten einen Wert vom Datentyp int (4 Byte) auf.
- Der Messbereich liegt zwischen 0 und 2000 ppm. (parts per million)
- Ein Jahr hat 365 Tage.
- Wie viele Daten sind in den 40 Jahren angefallen?

$$60 : 5 = 12$$

$$12 \cdot 24 = 288$$

$$288 \cdot 365 = \dots$$

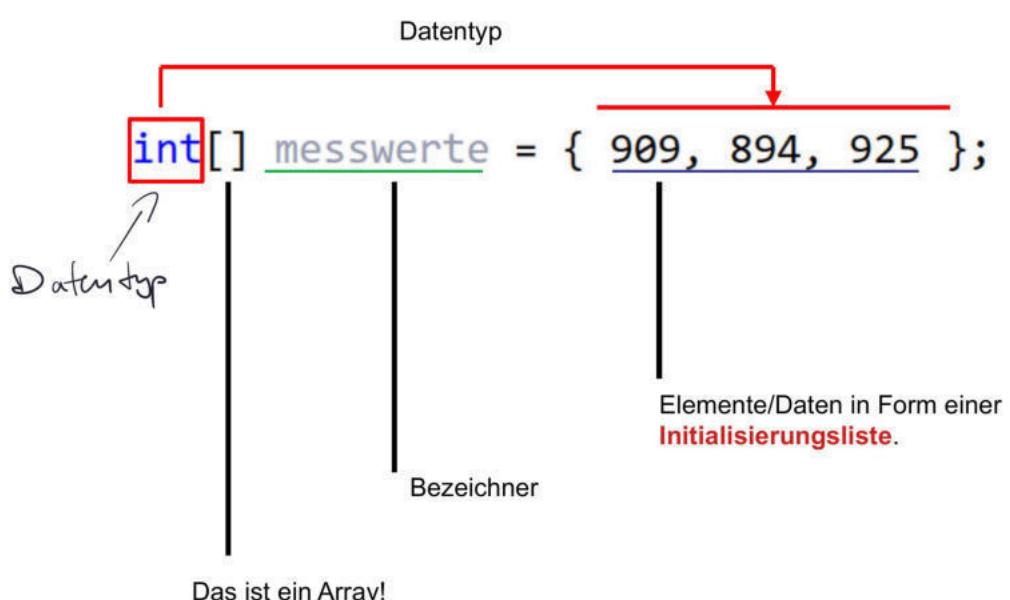
$$\dots \times 40 = 4.204.800 \approx 4 \cdot 10^6 \text{ Messwerte}$$

a 4 byte

Messung seit 40 Jahren...

- Das Gesundheitsamt möchte von der Schule den höchsten und den niedrigsten Wert mitgeteilt bekommen, der jemals im Klassenzimmer gemessen wurde.
- Wie würden Sie diese $4 \cdot 10^6$ Werte mit den Ihnen bekannten Mitteln in C# anlegen? (keine Datei!!!!)

Array mit Initialisierungsliste



Dieses Array hat die Länge 3.
Diese Integer Zahl erhält man über `messwerte.length`.

Arrays dimensionieren

- Wie kann man „leere“ Arrays anlegen?

Beispiel: Array mit 1000 Zeichen:

```
char[] Eingabe = new char[1000];
```

Annotations:

- ↑ *Array*
- ↑ *Datentyp*
- ↑ *1000 charakter*
- ↑ *lege an und reserviere im Rampeicher*

Arrays, Regeln Allgemein

- Die Elemente eines Arrays besitzen einen einzigen Datentyp.
- Arrays haben während ihrer Lebenszeit eine unabänderliche Größe.

```
int[] messwerte = { 909, 894, 925 };
```

Länge ist immer 3.

```
int[] messwerte = new int[1000];
```

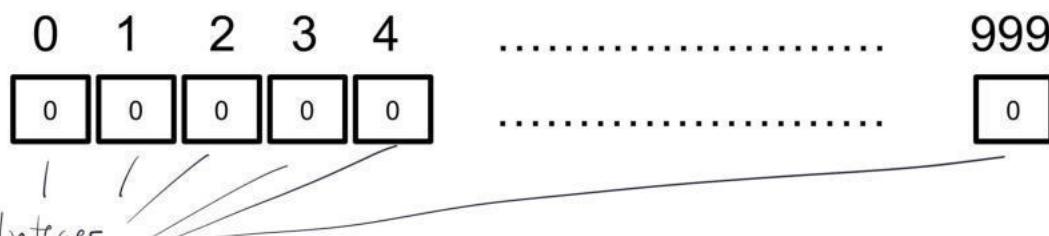
Länge ist immer 1000.

Arrays, Regeln Allgemein

- Die Elemente eines Arrays, das keine Initialisierungsliste hat, werden automatisch auf 0 (Null) initialisiert.

```
int[] messwerte = new int[1000];
```

- Aufbau des Arrays „Eingabe“:



- Arrays können alle Datentypen verwenden: float, integer, char usw. auch Klassen

Arrays, schreiben von Werten

- Die einzelnen Elemente eines Arrays können über einen Index (unsigned Integer) angesprochen werden.

```
int[] messwerte = new int[1000];
```

- Nach der Deklaration:



- Nach der Ausführung des Codes:

```
messwerte[0] = 5;  
messwerte[1] = 12;  
messwerte[2] = -42;  
messwerte[999] = 7;
```

} Integru „Platz“ einen Wert zuweisen.



Arrays, Lesen von Werten

- Die einzelnen Elemente eines Arrays können über einen Index (unsigned Integer) angesprochen werden.

```
char[] Eingabe = new char[1000];
```

Eingabe[0] = 'B';
Eingabe[1] = 'a';
Eingabe[2] = 'x';
Eingabe[999] = 'y';

char a = Eingabe[0];
char b = Eingabe[1];
char c = Eingabe[2];
char d = Eingabe[999];

Werte
geschrieben

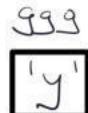
Werte
gelesen

Ausgabe in der Konsole:

B a x J

```
Console.WriteLine(a + " " + b + " " + c + " " + d);  
^ auch Eingabe[0] mögl.
```

- Inhalt:

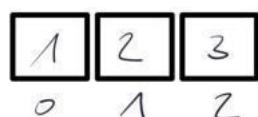


Tauschen von Element-Werten

- Gegeben ist folgendes Array:

```
int[] Daten = { 1, 2, 3 };
```

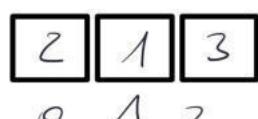
Inhalt:



- Tauschen Sie das Element an Stelle 0 mit dem Element an Stelle 1.

```
int t = Daten[0];  
Daten[0] = Daten[1];  
Daten[1] = t;
```

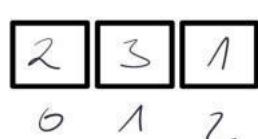
Inhalt:



- Tauschen Sie das Element an Stelle 1 mit dem Element an Stelle 2.

```
int t = Daten[1];  
Daten[1] = Daten[2];  
Daten[2] = t;
```

Inhalt:



Reflektion

- Auf welchen Wert werden die Elemente eines Arrays, ohne Initialisierungsliste, initialisiert?

0

- Kann man die Größe eines Arrays zur Laufzeit eines Programms ändern?

Nein

- Welche Datentypen kann man mit Arrays verwenden?

Alle

- Wie kann man Arrays der Reihe nach auslesen?

Mit Variablen Index (for-Schleife)

Corona-Fälle in Baden-Württemberg





- Veröffentlicht täglich eine Liste aller Corona Fälle in allen Kreisen/Städten in Baden-Württemberg.
- Leider ist die Menge der Zahlen unüberschaubar. Für eine Tageszeitung soll eine Software entwickelt werden, welche die Daten aufbereitet.
- Entwickeln Sie ein Programm welches die Stadt (den Kreis) mit der höchsten Anzahl an Infektionen findet und in der Konsole ausgibt.
- Ebenfalls soll die Stadt (der Kreis) mit der niedrigsten Anzahl an Infektionen ausgegeben werden.



MESK
Max-Eyth-Schule
Gewerbliche Schule Kirchheim unter Teck

Corona Fälle 2

Die Datengrundlage

Stadt/Landkreis	Fälle	Letzte Aktualisierung
Alb-Donau-Kreis	207	07.02.2023
Biberach	243	
Böblingen	396	
Bodenseekreis	245	
Breisgau-Hochschwarzwald	249	
Calw	231	
Emmendingen	197	
Enzkreis	317	
Esslingen	687	
Freudenstadt	192	
Göppingen	343	
Heidenheim	228	
Heilbronn	288	
Hohenlohekreis	155	
Karlsruhe	566	
Konstanz	366	
Lörach	349	
Ludwigsburg	643	
Main-Tauber-Kreis	136	
Neckar-Odenwald-Kreis	180	
Odenwaldkreis	697	
Ostalbkreis	496	
Rastatt	301	
Ravensburg	194	
Rems-Murr-Kreis	482	
Reutlingen	360	
Rhein-Neckar-Kreis	547	
Rottweil	235	
Schwäbisch Hall	292	
Schwarzwaldbaar-Kreis	305	
Sigmaringen	134	
Tübingen	217	
Tuttlingen	215	
Waldshut	259	
Zollernalbkreis	209	
Baden-Baden (Stadtkreis)	83	
Freiburg im Breisgau (Stadtkreis)	210	
Heidelberg (Stadtkreis)	82	



MESK
Max-Eyth-Schule
Gewerbliche Schule Kirchheim unter Teck

Sortieren mit dem Bubblesort

Wie gehen wir vor?

Array machen!

Die höchste/niedrigste Zahl ermitteln

Wie kommen wir an die Zahlen ran?

↳ mit for Schleife

Reflektion

- Wie kann man die Länge eines Arrays in C# ermitteln?

array-Name.Length;

- Wie kann man ein Array, Wert für Wert, untersuchen?

mit Schleife

- Ist es sinnvoll in einem C# Programm die Länge eines Arrays als bekannt anzunehmen?

Nein, weil sich die Länge ändern kann

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

namespace Schleifen_und_Arrays
{
    class Program
    {
        static void Main(string[] args)
        {
            float[] CoronaFaelle = { 207, 243, 396, 245, 249, 231, 197, 317, 687, 192, 343, 228, 288, 155, 566, 366,
349, 643, 136, 180, 697, 496, 301, 194, 482, 360, 547, 235, 292, 305, 134, 217, 215, 259, 209, 83, 210, 82,
207, 277, 414, 294, 593, 104 };
            string[] Kreis = {"Alb-Donau-Kreis", "Biberach", "Böblingen", "Bodenseekreis",
"Breisgau-Hochschwarzwald", "Calw", "Emmendingen", "Enzkreis", "Esslingen", "Freudenstadt",
"Göppingen", "Heidenheim", "Heilbronn", "Hohenlohekreis", "Karlsruhe", "Konstanz", "Lörrach",
"Ludwigsburg", "Main-Tauber-Kreis", "Neckar-Odenwald-Kreis", "Ortenaukreis", "Ostalbkreis", "Rastatt",
"Ravensburg", "Rems-Murr-Kreis", "Reutlingen", "Rhein-Neckar-Kreis", "Rottweil", "Schwäbisch Hall",
"Schwarzwald-Baar-Kreis", "Sigmaringen", "Tübingen", "Tuttlingen", "Waldshut", "Zollernalbkreis",
"Baden-Baden (Stadtkreis)", "Freiburg im Breisgau (Stadtkreis)", "Heidelberg (Stadtkreis)",
"Heilbronn (Stadtkreis)", "Karlsruhe (Stadtkreis)", "Mannheim (Stadtkreis)", "Pforzheim (Stadtkreis)",
"Stuttgart", "Ulm (Stadtkreis)" };

            float max = CoronaFaelle[0];
            float min = CoronaFaelle[0];
            string maxs=Kreis[0], mins=Kreis[0];
            for (int i=0;i<CoronaFaelle.Length;i++)
            {
                if (max < CoronaFaelle[i])
                {
                    max = CoronaFaelle[i];
                    maxs = Kreis[i];
                }
                if (min > CoronaFaelle[i])
                {
                    min = CoronaFaelle[i];
                    mins = Kreis[i];
                }
            }
            Console.WriteLine("Maximale Infektionszahl =" + max + " in " + maxs);
            Console.WriteLine("Minimale Infektionszahl = " + min + " in " +mins);
            Console.ReadKey();
        }
    }
}
```

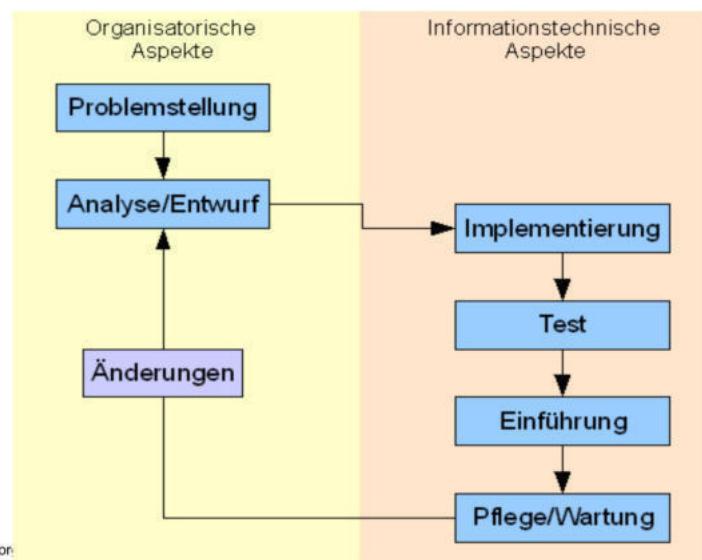
Thema



Vorgehensmodelle

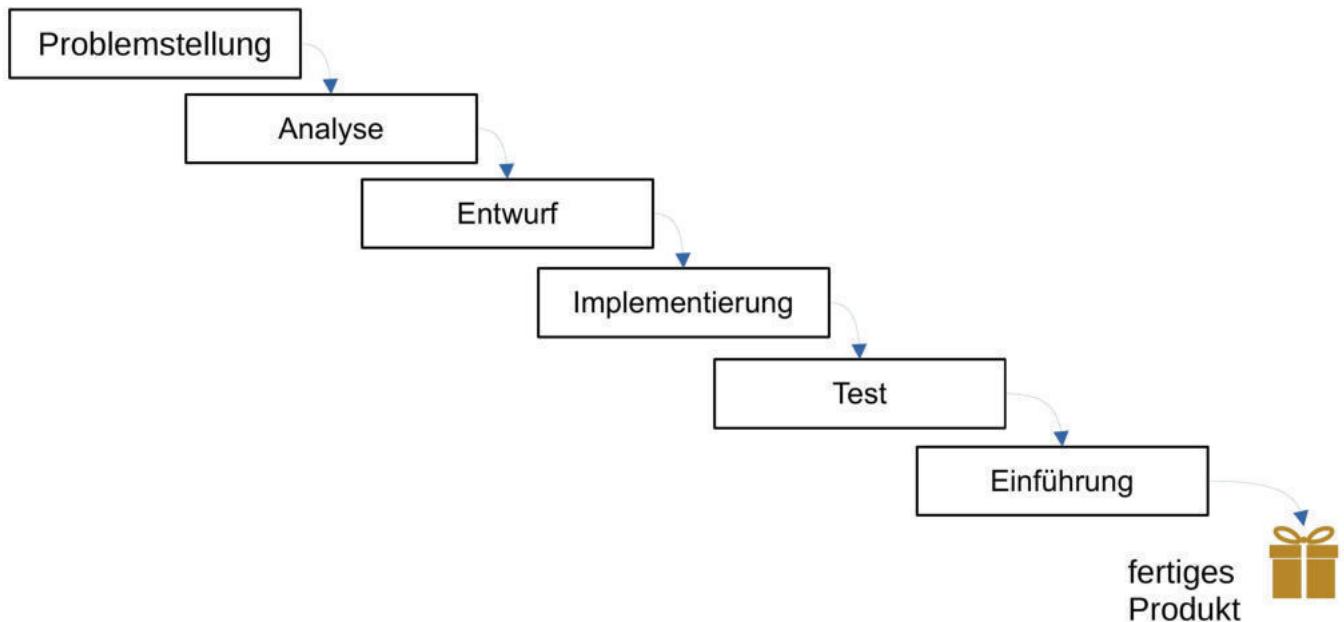
Der Software-Lebenszyklus

Der Software Lebenszyklus wird durch Vorgehensmodelle realisiert!



Quelle: Von Oliver Riedl - Eigenes Werk, Gemeinfrei, <https://commons.wikimedia.org>

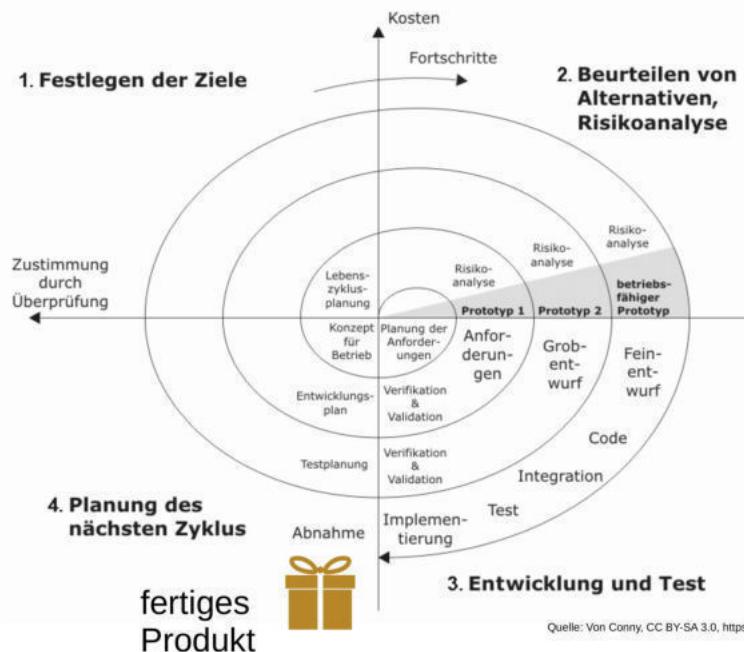
Wasserfallmodell



Wasserfallmodell

- Das Wasserfallmodell wird **linear** von links oben nach rechts unten abgearbeitet.
 - Im Wasserfallmodell fallen die Arbeitsergebnisse von einer Phase in die Nächste.
-
- + Klarer Entwicklungsablauf. Intuitiv anwendbar.
 - + Phasen sind klar bestimmbar.
 - + Kosten und Zeiten sind gut kalkulierbar *laut Modell*
 - Geringer Spielraum für Anpassungen aufgrund geänderter Anforderungen.
 - Suggeriert einen linearen Ablauf eines Softwareprojektes. Diesen gibt es in der Realität nicht.
 - Fehler wird erst am Ende des Entwicklungsprozesses erkannt.

Spiralmodell



Spiralmodell

- Das Spiralmodell ist ein iterativer Prozess in dem sich bestimmte Phasen während eines Projektes regelmäßig wiederholen.
- Das Spiralmodell ist eine Weiterentwicklung des Wasserfallmodells.

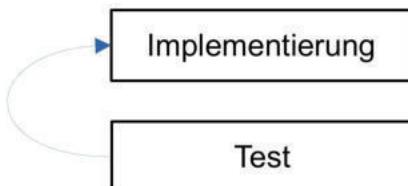
+ Anpassung an geänderte Anforderungen.

+ Phasen sind klar bestimmbar.

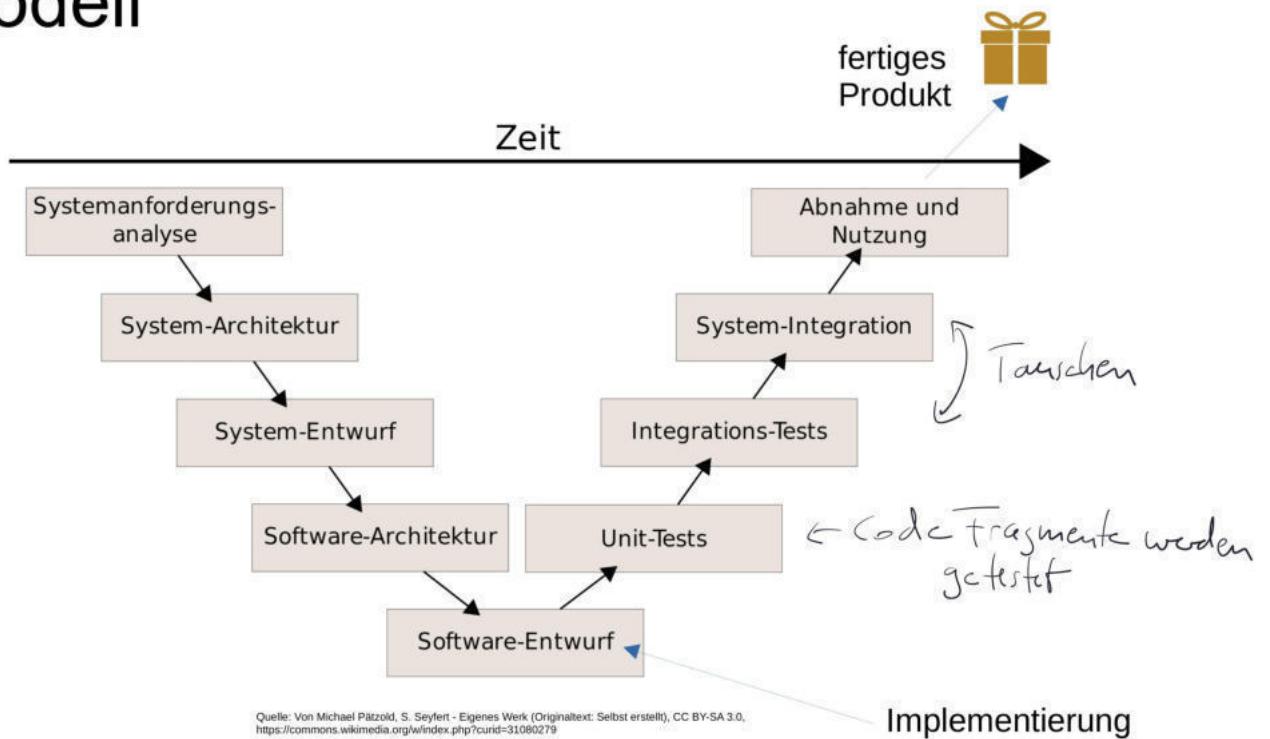
+ Kosten und Zeiten sind gut kalkulierbar.

- Risikobewertungen sind schwer durchzuführen.

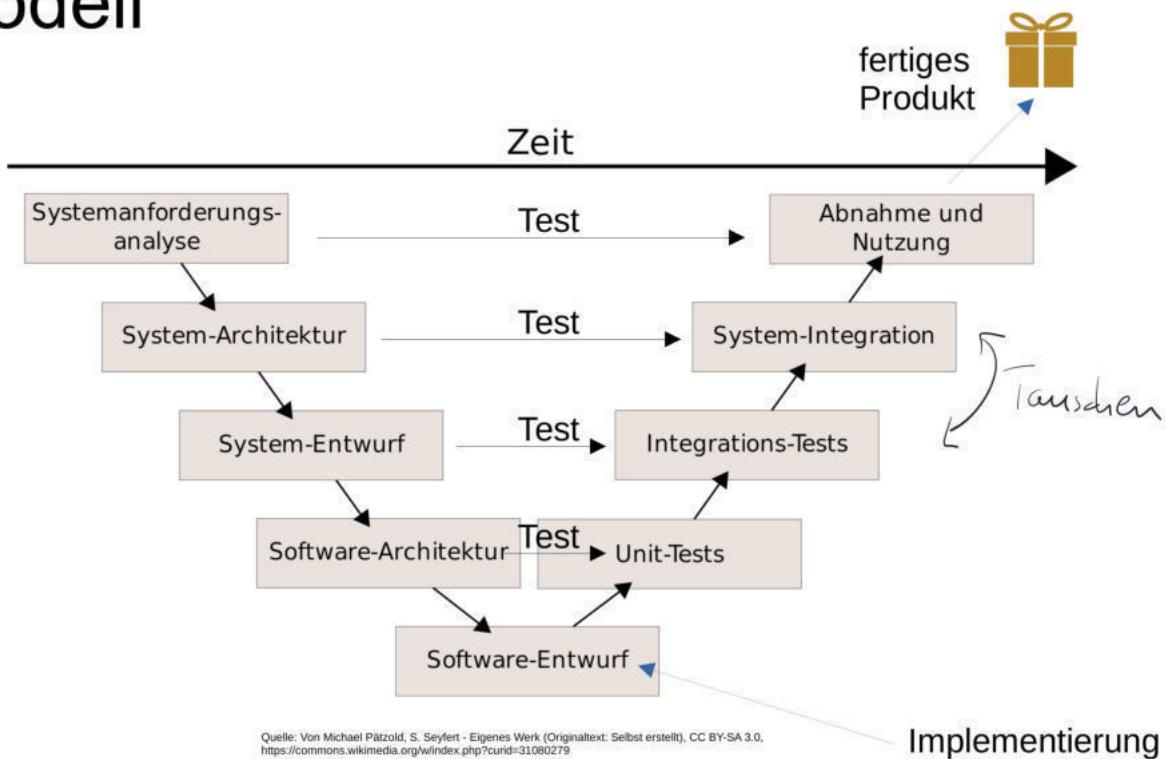
- Hoher Planungsaufwand.



V-Modell

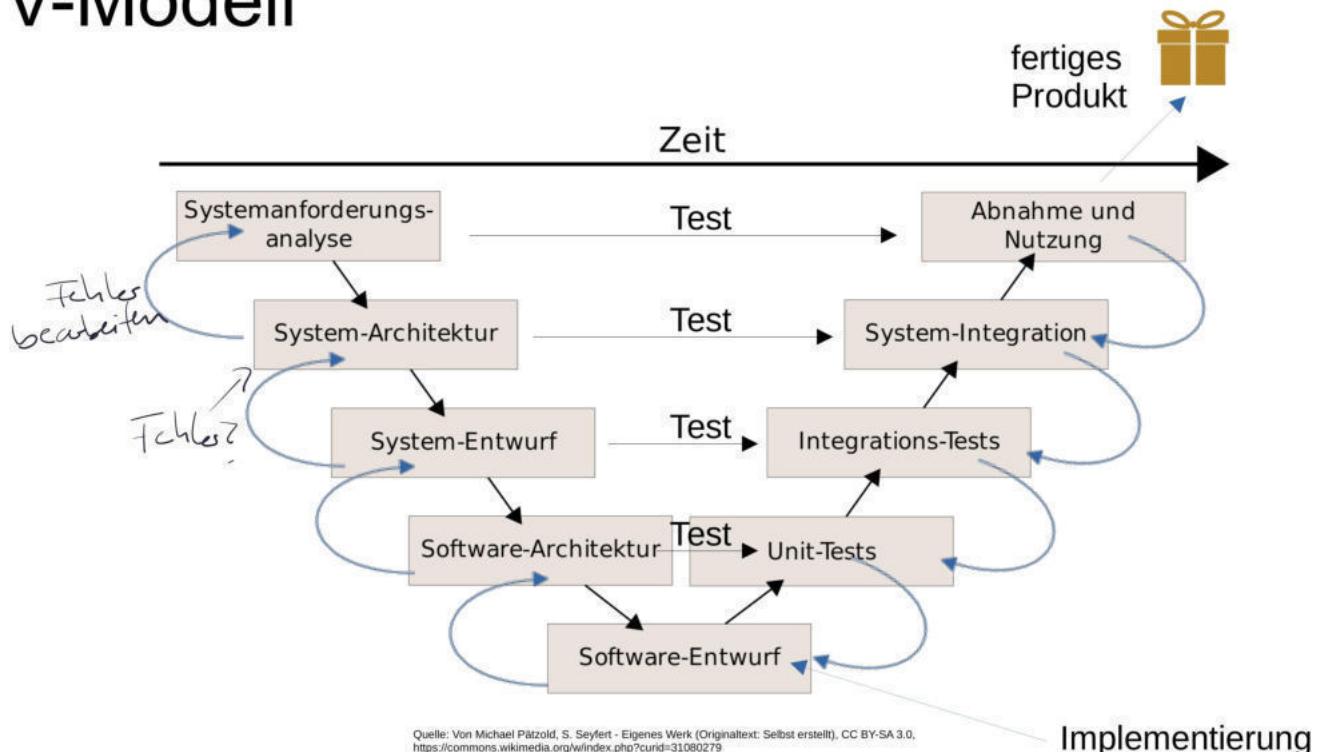


V-Modell



V-Modell

abstrakte Darstellung



Quelle: Von Michael Pätzold, S. Seyfert - Eigenes Werk (Originaltext: Selbst erstellt), CC BY-SA 3.0.
<https://commons.wikimedia.org/w/index.php?curid=31080279>

Implementierung

Gruppenarbeit

- Bilden Sie Gruppen von 10 Personen. Falls nicht möglich von 5 Personen.
- Falten Sie so viele Papierflieger, **vom Typ „Düse“**, wie möglich.
- Organisieren Sie sich nach dem Spiral-, Wasserfall- und V-Modell.

Gruppenarbeit

- Notieren Sie Ihre Beobachtungen/Ergebnisse:

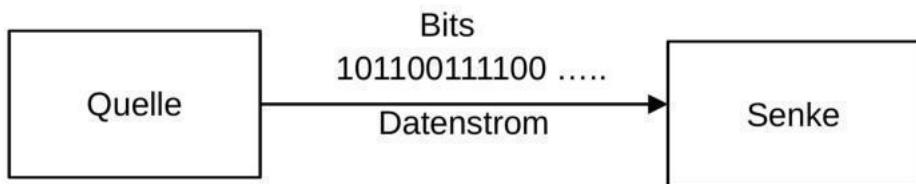
Vorgehensmodell Schwarm

Datenströme (Streams)



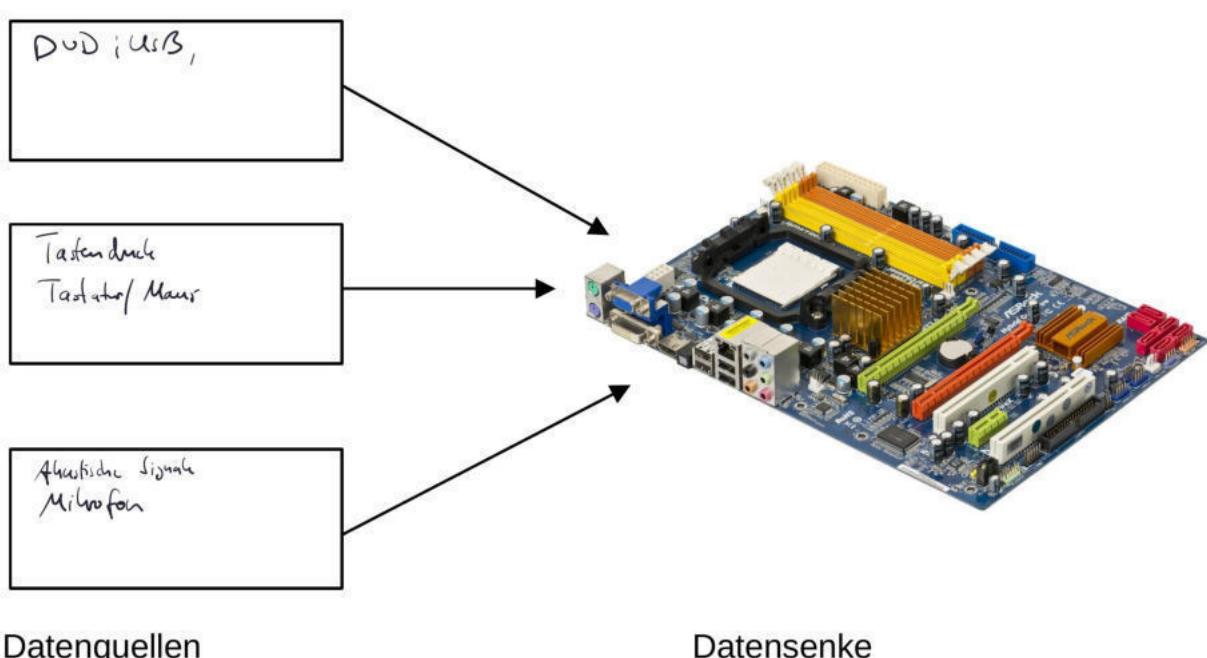
Was ist ein Datenstrom?

- Mit Datenströmen (englisch data streams) bezeichnet man einen Fluss von Daten die z.B. in Bits/Bytes unterteilt sind.
- Datenströme entspringen einer Datenquelle, sie ist der Datensender.
- Datenströme werden von einer Datensenke konsumiert, sie ist der Datenempfänger.



Beispiele für Datenquellen (nicht vollständig)

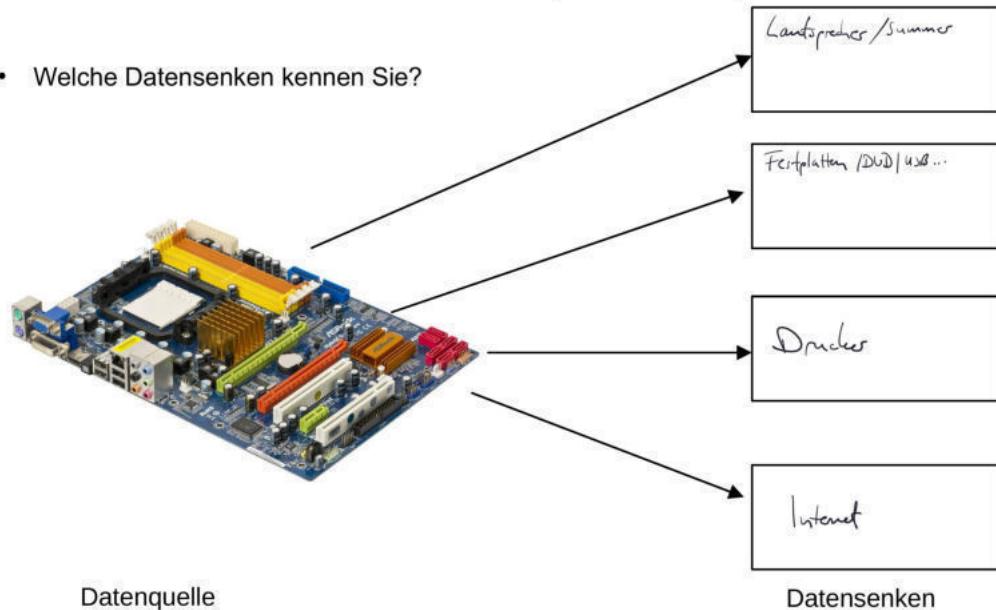
- Das Motherboard eines PCs ist in diesem Beispiel eine Datensenke.
- Welche Datenquellen kennen Sie?



Beispiele für Datensenken (nicht vollständig)

- Das Motherboard eines PCs ist in diesem Beispiel eine Datenquelle.

- Welche Datensenken kennen Sie?



4

Gemeinschaftlichkeit Datensenken und Datenquellen

- Welches Merkmal zeichnet alle digitalen Datenquellen aus?

→ Begrenzung bei Datenmenge und Datenübertragungsgeschwindigkeit

→ Man muss aktiv darauf (Die Quelle zugreifen)

Gemeinschaftlichkeit Datensenken und Datenquellen

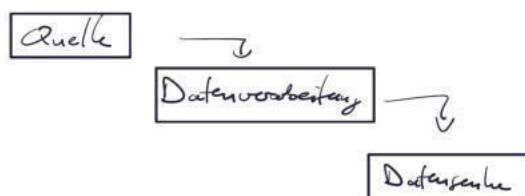
- Welches Merkmal zeichnet alle digitalen Datensenken aus?

→ Begrenzte Aufnahmekapazität

→ Begrenzte Übertragungsgeschwindigkeit

- Worin unterscheiden sich Datenquellen?

→ Die Datenrichtung
ist unterschiedlich

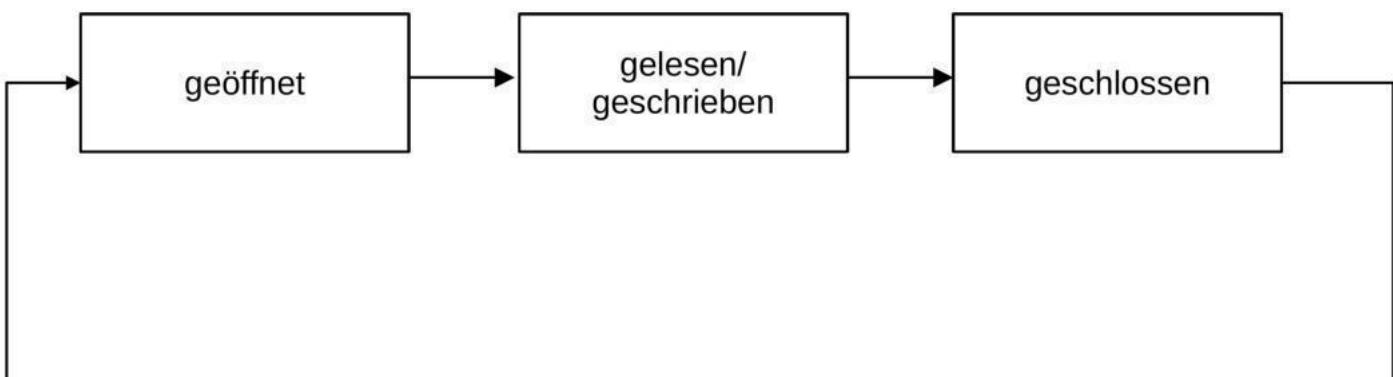


Vereinheitlichung

- Ziel einer Hochsprache ist es alle Datenquellen gleich zu behandeln.

- Datenquellen werden:

und Datensenken



Die Klasse Stream und ihre Kinder in C#

- Die Klasse Stream ist eine abstrakte Basisklasse.
- Sie stellt alle grundlegenden Eigenschaften und Methoden zum Lesen und Schreiben von Datenströmen bereit.

Close	→ Schließen eines Datenstroms.
Read	→ Lesen einer Folge von Bytes aus einem Datenstrom.
.ReadByte	→ Lesen eines Bytes aus einem Datenstrom.
Seek	→ Legt die aktuelle Position in einem Datenstrom fest.
Write	→ Schreibt eine Folge von Bytes in einen Datenstrom.
WriteByte	→ Schreibt ein Byte in einen Datenstrom.

- Öffnen eines Datenstroms bleibt den Kindklassen vorbehalten.
- Die Klasse „FileStream“ ist eine Kindklasse der Klasse Stream.
- Die Klasse „FileStream“ hat die Fähigkeit byteweise Dateien zu lesen und zu schreiben.

Beispiel, schreiben von Daten

```
using System;
using System.IO;

namespace FileStream_
{
    0 Verweise
    class Program
    {
        0 Verweise
        static void Main(string[] args)
        {
            byte[] arr = { 0x48, 0x61, 0x6c, 0x6c, 0x6f, 0x20, 0x6c, 0x69, 0x65, 0x62, 0x65, 0x20, 0x32, 0x42, 0x4b, 0x49, 0x31, 0x2e, 0xd, 0xa };

            FileStream fs = new FileStream("./Textdatei.txt", FileMode.Create);

            fs.Write(arr, 0, arr.Length);

            fs.Close();
        }
    }
}
```

Beispiel, lesen von Daten

```
class Program
{
    0 Verweise
    static void Main(string[] args)
    {
        FileStream fs = new FileStream("../.../TextFile1.txt", FileMode.Open);

        byte[] arr = new byte[fs.Length];

        fs.Read(arr, 0, (int) fs.Length);

        for (int i = 0; i < fs.Length; i++) Console.WriteLine("{0:X} ", arr[i]);

        Console.WriteLine("\n\n\n");

        for (int i = 0; i < fs.Length; i++) Console.Write((Char) arr[i]);

        fs.Close();
    }
}
```

Attribut FileMode

FileType	Beschreibung
FileMode.Append	Öffnet die Datei, sofern vorhanden, und schreibt an das Ende der Datei oder erstellt eine neue Datei.
FileMode.Create	Erstellen einer neuen Datei. Wenn die Datei bereits vorhanden ist, wird sie überschrieben.
FileMode.CreateNew	Erstellen einer neuen Datei. Wenn die Datei bereits vorhanden ist, wird ein Fehler als IOException-Ausnahme ausgelöst..
FileMode.Open	Vorhandene Datei öffnen. Eine FileNotFoundException-Ausnahme wird ausgelöst, wenn die Datei nicht vorhanden ist.
FileMode.OpenOrCreate	Datei öffnen, sofern diese vorhanden ist, oder eine neue Datei erstellen falls diese nicht vorhanden ist.
FileMode.Truncate	Eine vorhandene Datei öffnen. Wenn die Datei geöffnet wird, wird diese so abgeschnitten, dass deren Größe 0 Bytes beträgt.

Reflektion

- Welche Gemeinsamkeit haben alle Datenströme?
- Aus welchen Daten bestehen Datenströme?
- Welches Ziel hat man in einer Hochsprache bzgl. der Verarbeitung von Datenströmen?
- Warum wird beim Lesen eines Datenstroms in ein Array, die Größe des Arrays erst nach dem Öffnen des Datenstroms festgelegt?

Vergleichsoperatoren

Vergleichsoperator	Bedeutung
<code>==</code>	gleich
<code>!=</code>	ungleich
<code>></code>	größer
<code><</code>	kleiner
<code>>=</code>	größer gleich
<code><=</code>	kleiner gleich

Logische Operatoren (Datentyp „bool“)

Operator	Beschreibung
<code>!</code>	Negationsoperator --> Nicht
<code>&</code>	Und-Operator (erste Variante)
<code> </code>	Oder-Operator (erste Variante)
<code>^</code>	Xor-Operator
<code>&&</code>	Und-Operator (zweite Variante)
<code> </code>	Oder-Operator (zweite Variante)

Logische Operatoren (Datentyp „bool“)

Term 1	Term2	&&		^
false	false	false	false	false
true	false	false	true	true
false	true	false	true	true
true	true	true	true	false

Term	!
false	true
true	false

Ergänzen Sie die Tabellen.

Logische Operatoren (Datentyp „bool“)

```
int x;  
  
x=0;  
if ((x<5) && (x>1)) Console.WriteLine("Treffer!");  
x=1;  
if ((x<5) && (x>1)) Console.WriteLine("Treffer!");  
x=2;  
x=3;  
if ((x<5) && (x>1)) Console.WriteLine("Treffer!"); } }  
x=4;  
if ((x<5) && (x>1)) Console.WriteLine("Treffer!"); } }  
x=5;  
if ((x<5) && (x>1)) Console.WriteLine("Treffer!");  
x=6;  
if ((x<5) && (x>1)) Console.WriteLine("Treffer!");
```

Kreuzen Sie die Zeilen an in denen das Wort „Treffer“ ausgegeben wird.

Logische Operatoren (Datentyp „bool“)

```
using System;
namespace Operatoren
{
    class Program
    {
        static void Main(string[] args)
        {
            bool a=true,b=false,c=true,d=false;
            bool erg;
            Console.WriteLine("bool a=true,b=false,d=false,");
            erg = a && b;
            Console.WriteLine("a && b = " + erg);
            erg = b && a;
            Console.WriteLine("b && a = " + erg);
            erg = a && c;
            Console.WriteLine("a && c = " + erg);
            erg = c && a;
            Console.WriteLine("c && a = " + erg);
            erg = c && d;
            Console.WriteLine("c && d = " + erg);
            erg =d && c;
            Console.WriteLine("d && c = " + erg);

            erg = a ^ b;
            Console.WriteLine("a ^ b = " + erg);
            erg = b ^ a;
            Console.WriteLine("b ^ a = " + erg);
            erg = a ^ c;
            Console.WriteLine("a ^ c = " + erg);
            erg =c ^ a;
            Console.WriteLine("c ^ a = " + erg);
            erg = c ^ d;
            Console.WriteLine("c ^ d = " + erg);

            erg = !a;
            Console.WriteLine("!a = " + erg);
            erg = !c;
            Console.WriteLine("!c = " + erg);
        }
    }
}
```

Logische Operatoren (Datentyp „bool“)

Skizzieren Sie die Ausgabe des Programms der voran gegangenen Seite.

Bitweise Operatoren

Operator	Beschreibung
<code>~</code>	Negationsoperator --> Nicht
<code>&</code>	Und-Operator (erste Variante)
<code> </code>	Oder-Operator (erste Variante)
<code>^</code>	Xor-Operator
<code>&&</code>	Und-Operator (zweite Variante)
<code> </code>	Oder-Operator (zweite Variante)

C# Besonderheit: Das Ergebnis einer bitweisen Operation hat mindestens 32Bit!
Es ist deshalb vom Datentyp int oder uint oder long oder ulong

Bitweise Operatoren

```
namespace Operatoren
{
    class Program
    {
        static void Main(string[] args)
        {
            byte a=1,b=2,c=3,d=4;
            byte erg;
            Console.WriteLine("byte a=1,b=2,c=3,d=4;");
            erg = (byte) a & b;
            Console.WriteLine("a & b = " + erg);
            erg = (byte) a & c;
            Console.WriteLine("a & c = " + erg);
            erg = (byte) c & d;
            Console.WriteLine("a & d = " + erg);
            erg = (byte) a | b;
            Console.WriteLine("a | b = " + erg);
            erg = (byte) a | c;
            Console.WriteLine("a | c = " + erg);
            erg = (byte) c | d;
            Console.WriteLine("c | d = " + erg);
            erg = (byte) ~b;
            Console.WriteLine("~b = " + erg);
        }
    }
}
```

Warum wird ein Cast benötigt?

Bitweise Operatoren

Skizzieren Sie die Ausgabe des Programms der voran gegangenen Seite.

a=0000 0001_{bin} b=00000010_{bin} c=00000011_{bin} d=0000 0100_{bin}

a & b →	0000 0001 _{bin}	0000 0001 _{bin}
	<u>0000 0010_{bin}</u>	&&&& &&&&
	0000 0000 _{bin}	0000 0010 _{bin}
		0000 0000

Die Bitweisen Operatoren verknüpfen jedes Bit des einen Operanden mit seinem Pendant Bit im anderen Operanden.

In dem oberen Fall wird Bit Nummer 0 von a mit Bit Nummer 0 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 1 von a mit Bit Nummer 1 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 2 von a mit Bit Nummer 2 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 3 von a mit Bit Nummer 3 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 4 von a mit Bit Nummer 4 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 5 von a mit Bit Nummer 5 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 6 von a mit Bit Nummer 6 von b und verknüpft.
In dem oberen Fall wird Bit Nummer 7 von a mit Bit Nummer 7 von b und verknüpft.

Bitweise Operatoren

Skizzieren Sie die Ausgabe des Programms der voran gegangenen Seite.

a=0000 0001_{bin} b=00000010_{bin} c=00000011_{bin} d=0000 0100_{bin}

a & c →

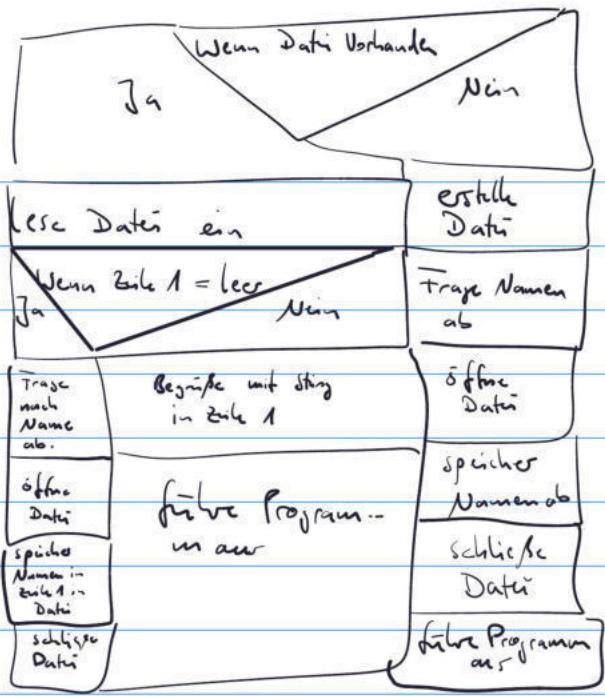
a & d →

a | b →

a | c →

c | d →

~b →



```

if (eingabe.Key == ConsoleKey.Delete)
{
    File.Delete("/home/fabi/Dokumente/C#/Projekte/Name_einlesen/Name.txt");
    Console.WriteLine("Name wurde gelöscht");
}
  
```

Welcher Operator Typ ist das?

```
public static void Main(string[] args)
{
    int a=7;      //0000 0000 0000 0111
    int b=5;      //0000 0000 0000 0101
    int c=8;      //0000 0000 0000 1000
    int ergebnis;

    ergebnis = a & b;
    Console.WriteLine(ergebnis);

    ergebnis = a & c;
    Console.WriteLine(ergebnis);

    ergebnis = a | b;
    Console.WriteLine(ergebnis);

    ergebnis = a | c;
    Console.WriteLine(ergebnis);

    ergebnis = a ^ c;
    Console.WriteLine(ergebnis);

    ergebnis = a ^ c;
    Console.WriteLine(ergebnis);

    ergebnis = b >> 1;
    Console.WriteLine(ergebnis);

    ergebnis = b >> 2;
    Console.WriteLine(ergebnis);

    ergebnis = b << 1;
    Console.WriteLine(ergebnis);

    ergebnis = b << 2;
    Console.WriteLine(ergebnis);
}
```

Ausgabe des Programms?

Welcher Operator Typ ist das?

```
public static void Main(string[] args)
{
    int a=7;
    int b=7;
    int c=8;

    if ((a>5) && (b>5))
    {
        Console.WriteLine("a");
    }
    if ((a<5) && (c==8))
    {
        Console.WriteLine("b");
    }

    if ((a>5) || (b>5))
    {
        Console.WriteLine("c");
    }
    if ((a<5) || (c==8))
    {
        Console.WriteLine("d");
    }
}
```

Ausgabe des Programms?

Bitweise Operatoren

Operator	Beschreibung
<code>~</code>	Negationsoperator --> Nicht
<code>&</code>	Und-Operator (erste Variante)
<code> </code>	Oder-Operator (erste Variante)
<code>^</code>	Xor-Operator
<code>&&</code>	Und-Operator (zweite Variante)
<code> </code>	Oder-Operator (zweite Variante)

Logische Operatoren

Operator	Beschreibung
!	Negationsoperator --> Nicht
&	Und-Operator (erste Variante)
	Oder-Operator (erste Variante)
^	Xor-Operator
&&	Und-Operator (zweite Variante)
	Oder-Operator (zweite Variante)

Vergleichsoperatoren

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

Operatoren



Aufgabe:

Auf dem Bildschirm erscheint diese Maske:

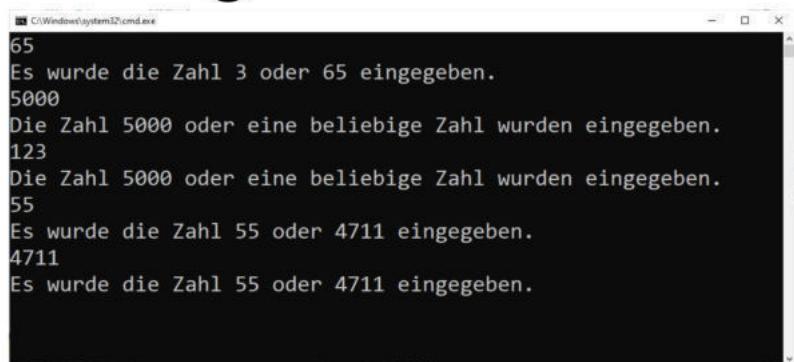
Geben Sie eine Zahl ein.

- 1: berechnet die Summe zweier Zahlen,
- 2: berechnet die Differenz zweier Zahlen,
- 3: berechnet das Produkt zweier Zahlen,
- 4: berechnet den Quotienten zweier Zahlen.
- sonst: Programmende

Bei Eingabe einer Zahl 1, 2, 3, 4 wird der entsprechende Programmteil aufgerufen (z.B. Summe zweier Zahlen)
Bei Eingabe einer anderen Zahl als 1, 2, 3, 4 wird das Programm beendet. Bitte C#-Programm dazu erstellen!

Switch-Case-Anweisung

```
static void Main(string[] args)
{
    while (true) //Endlosschleife Start
    {
        int zahl;
        zahl = Convert.ToInt32(Console.ReadLine());
        switch (zahl)
        {
            case 55:
            case 4711:
                Console.WriteLine("Es wurde die Zahl 55 oder 4711 eingegeben.");
                break;
            case 3:
            case 'A':
                Console.WriteLine("Es wurde die Zahl 3 oder 65 eingegeben.");
                break;
            case 5000:
            default:
                Console.WriteLine("Die Zahl 5000 oder eine beliebige Zahl wurden eingegeben.");
                break;
        }
    } //Endlosschleife Ende
```



```
C:\Windows\system32\cmd.exe
65
Es wurde die Zahl 3 oder 65 eingegeben.
5000
Die Zahl 5000 oder eine beliebige Zahl wurden eingegeben.
123
Die Zahl 5000 oder eine beliebige Zahl wurden eingegeben.
55
Es wurde die Zahl 55 oder 4711 eingegeben.
4711
Es wurde die Zahl 55 oder 4711 eingegeben.
```

Switch-Case-Anweisung

Es können in C# mehrere Case-Anweisungen deklariert werden.

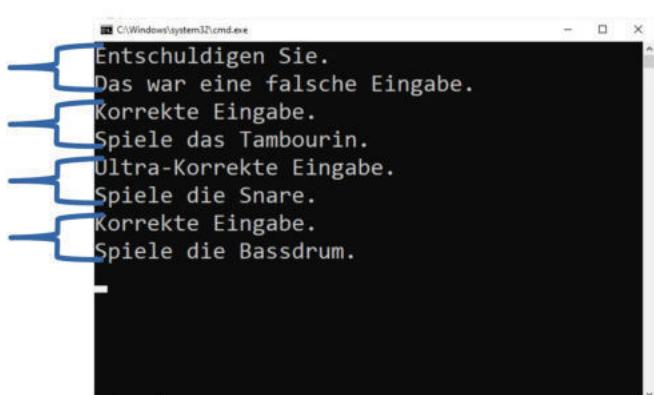
```
static void Main(string[] args)
{
    while (true) //Endlosschleife Start
    {
        int zahl;
        zahl = Convert.ToInt32(Console.ReadLine());
        switch (zahl)
        {
            case 55: ←
            case 4711: ←
                Console.WriteLine("Es wurde die Zahl 55 oder 4711 eingegeben.");
                break;
            case 3: ←
            case 'A': ←
                Console.WriteLine("Es wurde die Zahl 3 oder 65 eingegeben.");
                break;
            case 5000: ←
            default:
                Console.WriteLine("Die Zahl 5000 oder eine beliebige Zahl wurden eingegeben.");
                break;
        }
    } //Endlosschleife Ende
```

Switch-Case-Anweisung

Es können mehrere Anweisungen in einem Case-Block oder in einem Default-Block ausgeführt werden.

```
while (true) //Endlosschleife Start
{
    char eingabe;
    eingabe = Console.ReadKey(true).KeyChar;
    switch (eingabe)
    {
        case '1':
            Console.WriteLine("Korrekte Eingabe.");
            bassdrum.Play();
            Console.WriteLine("Spiele die Bassdrum.");
            break;
        case '2':
            Console.WriteLine("Korrekte Eingabe.");
            tambourin.Play();
            Console.WriteLine("Spiele das Tambourin.");
            break;
        case '3':
            Console.Write("Ultra-");
            Console.WriteLine("Korrekte Eingabe.");
            snare.Play();
            Console.WriteLine("Spiele die Snare.");
            break;
        default:
            Console.WriteLine("Entschuldigen Sie.");
            Console.WriteLine("Das war eine falsche Eingabe.");
            break;
    }
}
```

Beispiel
Eingabe:



Switch-Case-Anweisung

Programmablauf wenn eingabe = `2` , d.h. eingabe = 50_{dez}

```
eingabe = '2';
switch (eingabe)
{
    case '1':
        bassdrum.Play();
        break;
    case '2':
        tambourin.Play();
        break;
    case '3':
        snare.Play();
        break;
    default:
        Console.WriteLine("Falsche Eingabe");
        break;
}
```

Technisch gesehen, ist die break;-Anweisung ein Sprung hinter das Ende des Switch-Blockes.

Switch-Case-Anweisung

```
switch (eingabe)
{
    case '1':
        bassdrum.Play();
        break; ◀
    case '2':
        tambourin.Play();
        break; ◀
    case '3':
        snare.Play();
        break; ◀
    default:
        Console.WriteLine("Falsche Eingabe");
        break; ◀
}
```

break; = Verlassen der SWITCH-Anweisung.

Switch-Case-Anweisung

In C, C++, Java usw. ein ganzzahliger Ausdruck.
In C# sind Strings, Fließkommazahlen usw. auch zulässig.

Konstante

```
switch (eingabe)
{
    case '1':
        bassdrum.Play();
        break;
    case '2':
        tambourin.Play();
        break;
    case '3':
        snare.Play();
        break;
    default: ◀
        Console.WriteLine("Falsche Eingabe");
        break;
}
```

Wenn kein Wert passt.

Switch-Case-Anweisung

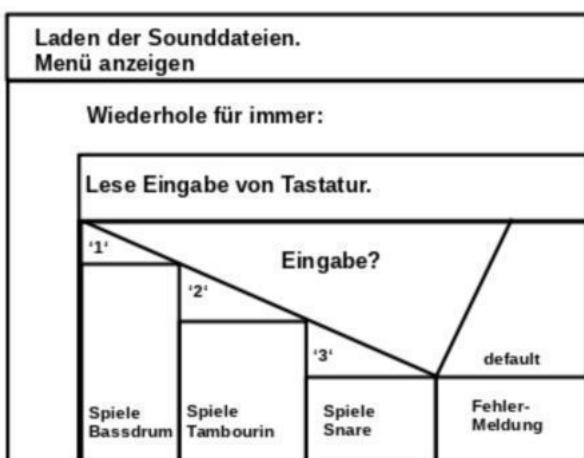
Anweisung(en).

```
switch (eingabe)
{
    case '1':
        ► bassdrum.Play();
        break;
    case '2':
        ► tambourin.Play();
        break;
    case '3':
        ► snare.Play();
        break;
    default:
        ► Console.WriteLine("Falsche Eingabe");
        break;
}
```

Ein switch ist ein Schalter, der verschiedene Zustände (wie bei einer Gangschaltung) haben kann.
Hier sind die Zustände '1','2','3'.

Beispiel

- Auf die Tasteneingaben 1,2 und 3 sollen unterschiedliche Trommelklänge abgespielt werden. Für jede andere Tasteneingabe wird eine Fehlermeldung ausgegeben.
- Zeichnen Sie das Struktogramm der Mehrfachauswahl.



```
static void Main(string[] args)
{
    //Klänge in den RAM-Speicher laden
    SoundPlayer bassdrum = new SoundPlayer(@"../../bassdrum.wav");
    SoundPlayer tambourin = new SoundPlayer(@"../../tambourin.wav");
    SoundPlayer snare = new SoundPlayer(@"../../snare.wav");
    while (true) //Endlosschleife Start
    {
        char eingabe;
        eingabe = Console.ReadKey(true).KeyChar;
        switch (eingabe)
        {
            case '1':
                bassdrum.Play();
                break;
            case '2':
                tambourin.Play();
                break;
            case '3':
                snare.Play();
                break;
            default:
                Console.WriteLine("Falsche Eingabe");
                break;
        } //Endlosschleife Ende
    }
}
```

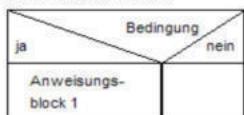
Struktogrammblöcke

Anweisungen

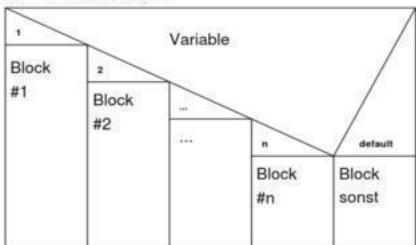


Kontrollstrukturen

Zweifachauswahl



Mehrfachauswahl



Schleifen

Zählergesteuerte Schleife



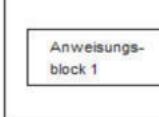
Kopfgesteuerte Schleife



Fußgesteuerte Schleife



Endlosschleife



Thema

Kontrollstrukturen V

Mehrfachauswahl

Switch-Case

Anweisung

Beispiel 3. Zahlenrätsel, Rechenzeit

- Code, zur Bestimmung der Rechenzeit.
→ Benötigt auf einem 3GHz Rechner ca. 20 Sekunden.

```
static void Main(string[] args)
{
    int s, e, n, d, m, o, r;
    int send, more, money;
    Console.ReadKey();
    Console.WriteLine("Start");
    for (int xx=0;xx<999;xx++)
    for (s = 0; s <= 10; s++)
    {
        for (e = 0; e <= 10; e++)
        {
            for (n = 0; n <= 10; n++)
            {
                for (d = 0; d <= 10; d++)
                {
                    for (m = 0; m <= 10; m++)
                    {
                        for (o = 0; o <= 10; o++)
                        {
                            for (r = 0; r <= 10; r++)
                            {
                                send = 1000 * s + 100 * e + 10 * n + d;
                                more= 1000 * m + 100 * o + 10 * r + e;
                                money = send + more;
                                //Console.WriteLine("send \t" + send);
                                //Console.WriteLine("more \t" + more);
                                //Console.WriteLine("-----");
                                //Console.WriteLine("money \t" + money);
                                //Console.WriteLine(); Console.WriteLine();
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Beispiel 3. Zahlenrätsel, Rechenzeit

- Textausgaben sind „teuer“, d.h. sie benötigen viel Rechenleistung.
→ Textausgaben auskommentieren.
- Zeit mit Stoppuhr (z.B. Uhr am Handgelenk) messen.
- Wenn Zeit zu kurz, Schleifen mehrfach durchlaufen lassen.

Beispiel 3. Zahlenrätsel

- 7 Fach verschachtelte Schleife

```
static void Main(string[] args)
{
    int s, e, n, d, m, o, r;
    int send, more, money;
    Console.ReadKey();
    Console.WriteLine("Start");
    for (s = 0; s <= 10; s++)
    {
        for (e = 0; e <= 10; e++)
        {
            for (n = 0; n <= 10; n++)
            {
                for (d = 0; d <= 10; d++)
                {
                    for (m = 0; m <= 10; m++)
                    {
                        for (o = 0; o <= 10; o++)
                        {
                            for (r = 0; r <= 10; r++)
                            {
                                send = 1000 * s + 100 * e + 10 * n + d;
                                more = 1000 * m + 100 * o + 10 * r + e;
                                money = send + more;
                                Console.WriteLine("send \t " + send);
                                Console.WriteLine("more \t " + more);
                                Console.WriteLine("-----");
                                Console.WriteLine("money \t " + money);
                                Console.WriteLine(); Console.WriteLine();
                            }
                        }
                    }
                }
            }
        }
    }
}
Console.WriteLine("Fertig");
Console.ReadKey();
```



Beispiel 3. Zahlenrätsel

- Die Variablen s, e, n, d, m, o, r, e, m, o, n, e, y können jeweils Werte von 0 bis 9 annehmen.
- Es sollen alle Kombinationen gefunden werden die folgendem Muster entsprechen:

$$\begin{array}{r} \text{send} \\ + \text{more} \\ \hline \text{money} \end{array}$$

$$\begin{array}{r} \text{send} \\ + \text{more} \\ \hline \text{money} \end{array}$$

1 2 3 4
5 6 7 2
5 6 3 2 8

Beispiel 2 Verschachtelte Schleifen

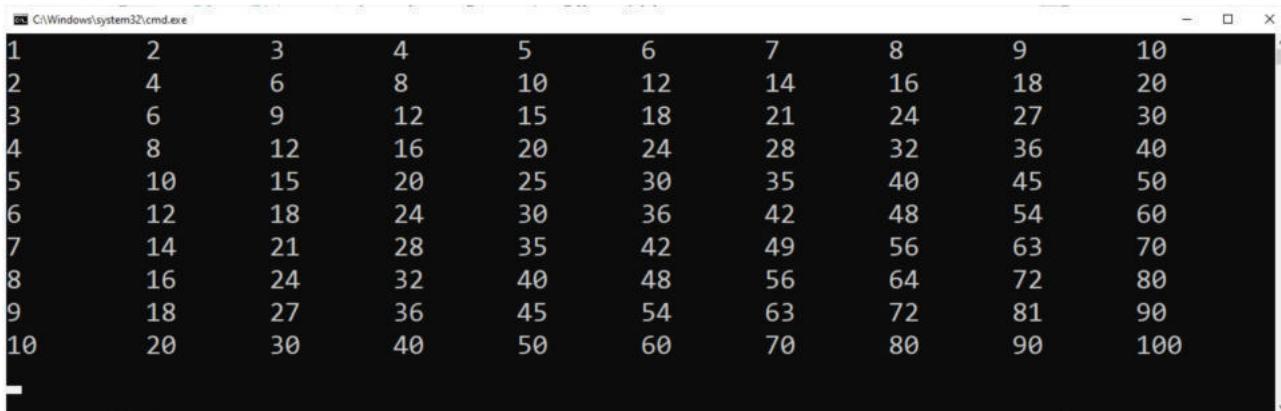
- Der Code:

Beispiel 2 Verschachtelte Schleifen

- Das Struktogramm:

Beispiel 2 Verschachtelte Schleifen

- Die Ausgabe in der Konsole.



The screenshot shows a Windows command prompt window titled 'C:\Windows\system32\cmd.exe'. It displays a 10x10 multiplication table where each row and column index is followed by its corresponding values. The table starts with 1x1=1 and ends with 10x10=100.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Beispiel 2 Verschachtelte Schleifen

- Schreiben Sie ein Programm, das die folgende, hellblau gekennzeichnete Multiplikationstabelle erstellt.

	*	1	2	3	4	...	10
1	1	2	3	4	...	10	
2	2	4	6	8	...	20	
3	3	6	9	12	...	30	
4	4	8	12	16	...	40	
...	
10	10	20	30	40	...	100	

Beispiel 1. Summenformel

- Entwickeln Sie das C# Programm:

Beispiel 1. Summenformel

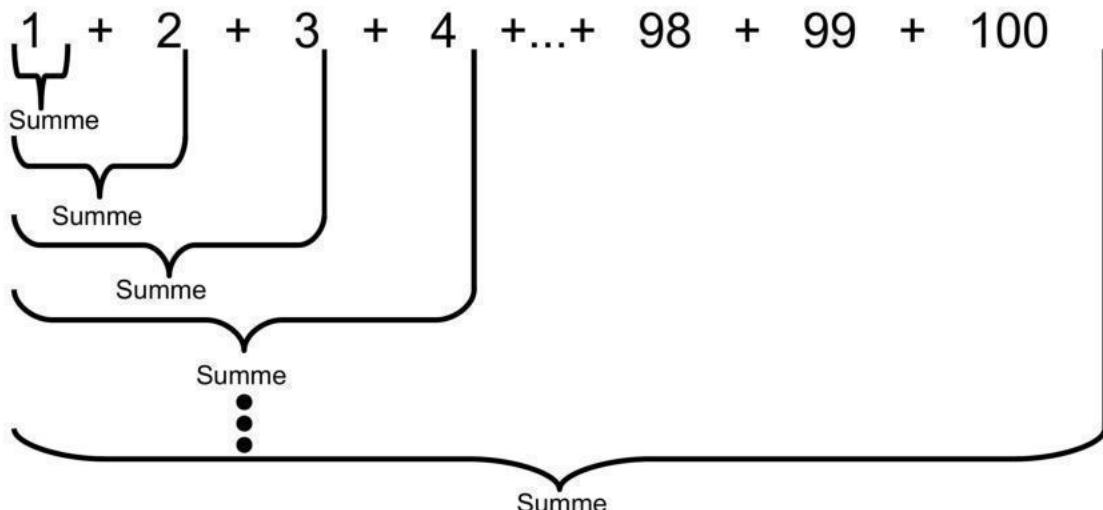
- Zeichnen Sie das Struktogramm.

Beispiel 1, Summenformel

- Schreiben Sie ein Programm (auf Papier), das die folgende Summe berechnet:

$$1+2+3+4+\dots+98+99+100$$

- Idee:



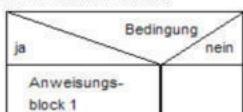
Struktogrammblöcke

Anweisungen

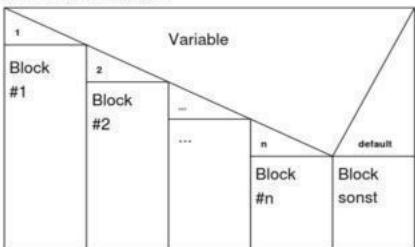


Kontrollstrukturen

Zweifachauswahl

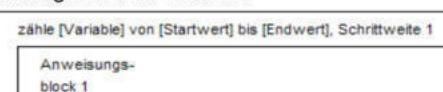


Mehrfachauswahl



Schleifen

Zählergesteuerte Schleife



Kopfgesteuerte Schleife



Fußgesteuerte Schleife



Endlosschleife



Thema

Schleifen II



Aufgabe Continue

- Skizzieren Sie die Ausgabe des folgenden Codes.

```
for (i=0;i<=5;i++)  
{  
    If (i>2) continue;  
    Console.WriteLine(i);  
}  
  
Console.WriteLine("I ist"+ i);
```

Aufgabe Break

- Skizzieren Sie die Ausgabe des folgenden Codes.

```
for (i=0;i<=5;i++)  
{  
    If (i==2) break;  
    Console.WriteLine(i);  
}  
  
Console.WriteLine("I ist "+ i);
```

Break und Continue

Für alle drei Arten von Schleifen, die For-Schleife, die While-Schleife und die Do-While-Schleife existieren zwei wichtige Anweisungen, **break** und **continue**.

Die **break**-Anweisung veranlasst, dass die Schleife sofort verlassen wird.
Bei verschachtelten Schleifen wird die aktuelle Schleife sofort verlassen.

Die **continue**-Anweisung veranlasst, dass sofort zum Anfang der Schleife (Schleifen-Bedingung) gesprungen wird. Im Gegensatz zur break-Anweisung wird die Schleife nicht verlassen.

Aufgabe 3

- Skizzieren Sie die Ausgabe des folgenden Codes.

```
for (i=0;i<=5;i++) Console.WriteLine(i);
```

Aufgabe 3

- Skizzieren Sie die Ausgabe des folgenden Codes.

```
for (i=0;i<=5;i++) Console.WriteLine(i);
```

Aufgabe 2

- Skizzieren Sie die Ausgabe des folgenden Codes.

```
for (i=0;i<=5;i++) {  
    Console.WriteLine(i);  
}
```

Aufgabe 1

- Für einen Raketenstart soll ein Computer von 10 bis 0 zählen. Bei 0 soll die Rakete starten.

Initialisierungen

Zähle i von bis , Schrittweite 1

Ausgabe i

Spreche Zahl i

Ausgabe „Start“
Spreche „Start“

```
static void Main(string[] args)  
{  
    int i;  
    SpeechSynthesizer synth = new SpeechSynthesizer();  
    synth.SetOutputToDefaultAudioDevice();  
  
    for (  
    {  
        Console.WriteLine(i);  
        synth.Speak(Convert.ToString(i));  
    }  
    Console.WriteLine("Start");  
    synth.Speak("Start");  
    Console.ReadKey();  
}
```

Ausführung For-Schleife, Beispiel

```
for (i=0;i<=5;i++)  
{  
    Console.WriteLine(i);  
}  
1  
2  
3  
4  
5
```

Ausführung For-Schleife, Beispiel

```
using System;  
public class Program  
{  
    public static void Main()  
    {  
        int i=0;  
        for (Console.WriteLine("Anweisung 1");i<10;Console.WriteLine("anweisung 2"))  
        {  
            Console.WriteLine("Hello World");  
        }  
    }  
}
```

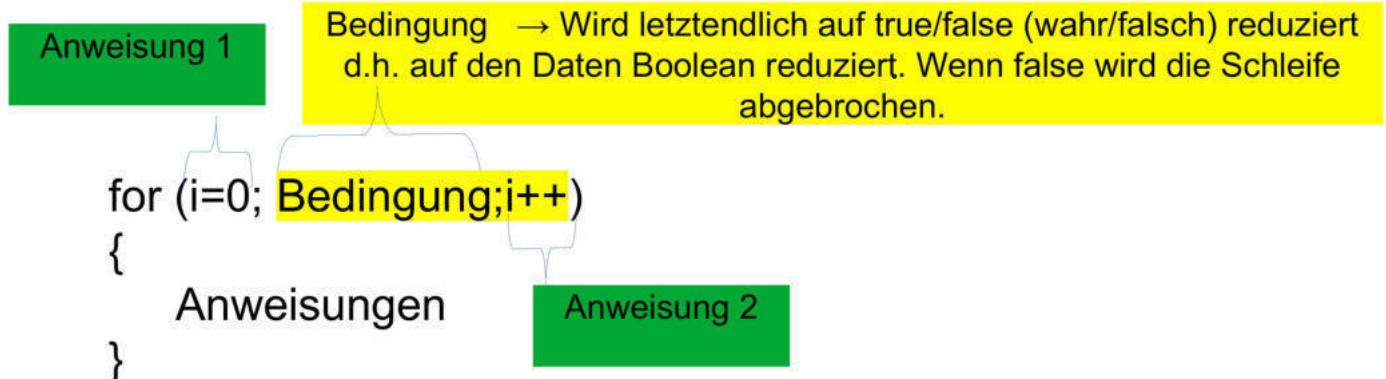
Anweisung 1
Hello World
anweisung 2
Hello World
.

Ausführung For-Schleife

Ausführungsreihenfolge:

X	X
for (A; B; C)	A
{	B
D	D
}	C
E	B
F	D
	.
	.
	.
	C
	B → Wenn Bedingung == false
	E
	F

Aufbau For-Schleife



Anweisung 1: Wird einmalig vor dem **ersten** Durchlauf der Schleife ausgeführt.
Wird, für gewöhnlich zur einmaligen Initialisierung eines Schleifenzählers verwendet.

Anweisung 2: Wird ab dem **zweiten** Schleifendurchlauf bei jedem folgenden Schleifendurchlauf ausgeführt. Wird gewöhnlich zur Veränderung des Schleifenzählers verwendet.

Beispiel

- Für einen Raketenstart soll ein Computer von 0 bis 10 zählen. (Bei 10 soll die Rakete starten.)

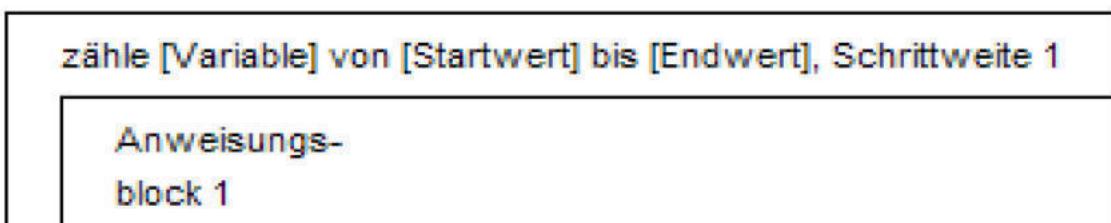


```
static void Main(string[] args)
{
    int i;
    SpeechSynthesizer synth = new SpeechSynthesizer();
    synth.SetOutputToDefaultAudioDevice();

    for (i = 0; i <= 10; i++)
    {
        Console.WriteLine(i);
        synth.Speak(Convert.ToString(i));
    }
    Console.WriteLine("Start");
    synth.Speak("Start");
    Console.ReadKey();
}
```

Struktogramm Strukturblock

- Der Strukturblock ist dem der While-Schleife sehr ähnlich.



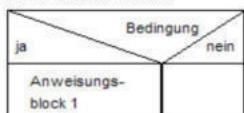
Struktogrammblöcke

Anweisungen

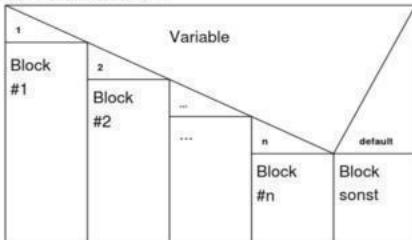


Kontrollstrukturen

Zweifachauswahl



Mehrfachauswahl



Schleifen

Zählergesteuerte Schleife

```
zähle [Variable] von [Startwert] bis [Endwert], Schrittweite 1  
Anweisungs-block 1
```

Kopfgesteuerte Schleife

```
solange Bedingung wahr  
Anweisungs-block 1
```

Fußgesteuerte Schleife

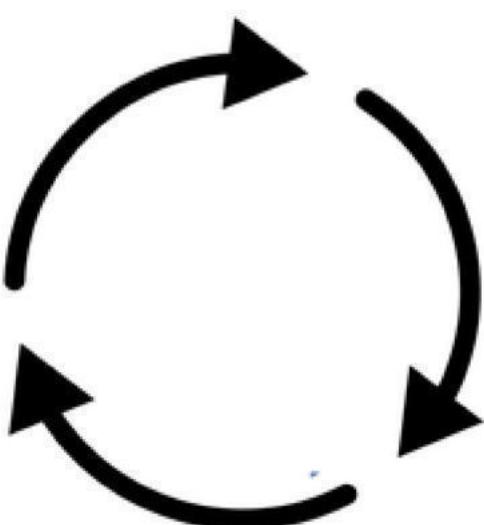
```
Anweisungs-block 1  
solange Bedingung wahr
```

Endlosschleife

```
Anweisungs-block 1
```

Thema

Schleifen



^

<u>Typ</u>	<u>Speicherplatz</u>	<u>Wertebereich</u>
byte	1 Byte	0 ... 255
sbyte	1 Byte	-128 ... 127
char	2 Byte	0 ... 65535
int	4 Byte	$-2^{31} \dots +2^{31}-1$
short	2 Byte	$-2^{15} \dots +2^{15}-1$
long	8 Byte	$-2^{63} \dots +2^{63}-1$
float	4 Byte	$\pm 1.5 \times 10^{-45}$ bis $\pm 3,4 \times 10^{38}$
double	8 Byte	$\pm 5.0 \times 10^{-324}$ bis $\pm 1,7 \times 10^{308}$
decimal	16 Byte	$\pm 1.0 \times 10^{-28}$ bis $\pm 7,9 \times 10^{28}$

Aufgabe 4:

a) Welchen Wert hat die Variable i nach Beendigung der While-Schleife?

10P

```
short i = 32767;  
while (i>0)  
{  
    i++;  
}  
Console.WriteLine(i);
```

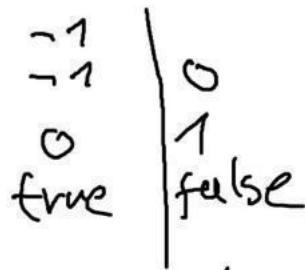
Bis die Zahl negativ
wird.

-32768

b) Welchen Wert hat die Variable t nach Beendigung der Do-While-Schleife?

10P

```
sbyte t = -1;  
do  
{  
    t++;  
} while (t < 1);  
Console.WriteLine(t);
```



1 1

Kontrollstrukturen

Aufgabe 3 :

a) Zeichnen Sie den Struktogramm Strukturblock der IF-THEN-ELSE Anweisung.

5P

b) Welche Anweisung wird ausgeführt. Kreuzen Sie an.

```

int eingabe = 1; 
int code = 1;   

if (eingabe == code) Anweisung_a(); 
else Anweisung_b();   

if (eingabe == code) Anweisung_a(); 
if (eingabe != code) Anweisung_b();   

if (eingabe == code)
    Anweisung_a(); 
    Anweisung_b();  Extra code
if (eingabe != code) Anweisung_c(); 
Anweisung_d(); 

```

Typkonvertierung

Aufgabe 2 :

a) In einem C# Programm befindet sich folgender Code.

2P

```
int a = 123;  
float b;  
b = a;
```

Welchen Namen hat diese Art der Konvertierung?

b) Korrigieren Sie folgendes Programm:

2P

```
static void Main(string[] args)  
{  
    int a;  
    float b = 123;  
    a = b;  
}
```

c) Wie bezeichnet man diese Art der Konvertierung?

2P

```
byte x;  
int y=32767;  
x = (byte) y;
```

Grundlagen

Aufgabe 1:

a) Worin unterscheiden sich Fließkommazahlen von Integerzahlen? 2P

b) Berechnen Sie den Wertebereich einer 6-Bit Zahl. (Annahme Wertebereich beginnt bei 0 und wächst in Einerschritten.) 2P

c) Das EVA Prinzip. Ordnen Sie Zu Eingabe, Verarbeitung, Ausgabe: 3P



d) Nennen Sie zwei Ganzzahlige Datentypen. 2P

Pseudocode

- Implementierung

 Max-Eyth-Schule Gewerbliche Schule Kirchheim unter Teck	Klassenarbeit PROG-T 2BKI1, Datum: 26.11.2021	Name: Note: Punkte:
---	---	---------------------------

Grundlagen

Aufgabe 1:

a) Worin unterscheiden sich Fließkommazahlen von Integerzahlen?

b) Berechnen Sie den Wertebereich einer 6-Bit Zahl. (Annahme Wertebereich beginnt bei 0 und wächst in Einerschritten.)



Pseudocode

- Der Pseudocode ist eine Programmiersprache.
- Sie dient zur Dokumentation von Algorithmen.
- Pseudocode ist eine Sprache ohne Maschinensprache.
- Es gibt keine Laufzeit.

c) Das EVA Prinzip. Ordnen Sie Zu Eingabe, Verarbeitung, Ausgabe:



d) Nennen Sie zwei Ganzzahlige Datentypen.

Beispiel:

```
WHILE (S  
IF (letz  
{I  
ELSE  
IF
```



Struktogramm

Groove implementieren Teil 2

- Ergänzen das Struktogramm um eine DO-While Schleife, die den Grundrhythmus 3x wiederholt.

Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.

Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.

Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.

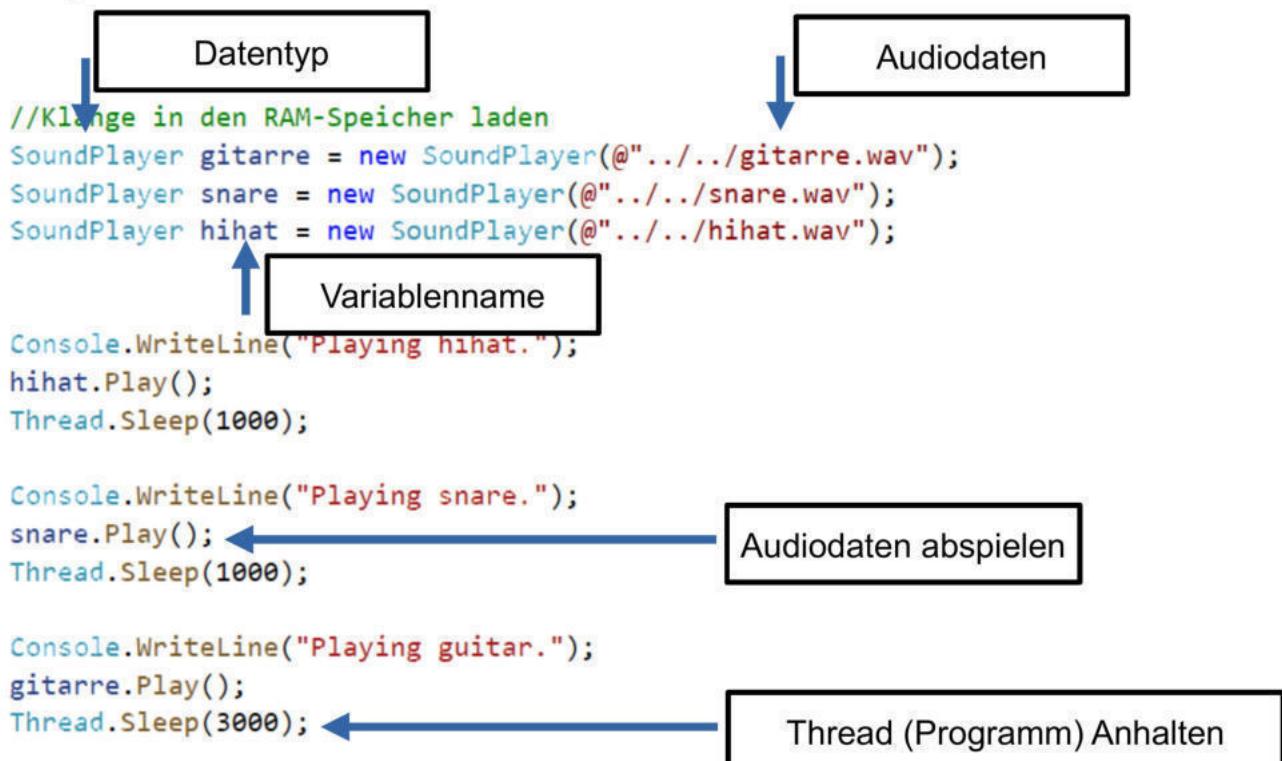
Groove implementieren Teil 1

- Die zu spielende Sequenz besteht aus folgendem Grundrhythmus:

Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.

- Verwenden Sie zur Wiederholung des Hihat-Sounds eine DO-While Schleife.
- Zeichnen Sie das Struktogramm auf Seite 8.
- Positionieren Sie das Struktogramm in der Mitte, da dieses in einem zweiten Schritt erweitert wird.

Vorlage



Training: Drum Sounds

- Sie sollen für ein Intro eines Computerspiels einen einfachen Groove programmieren.

- Da Ihr Chef für eine Vorführung auf eine Messe ein schnelle Umsetzung benötigt, sollen Sie den Groove in C# implementieren.

- Im Ergebnis soll diese Sequenz abgespielt werden:

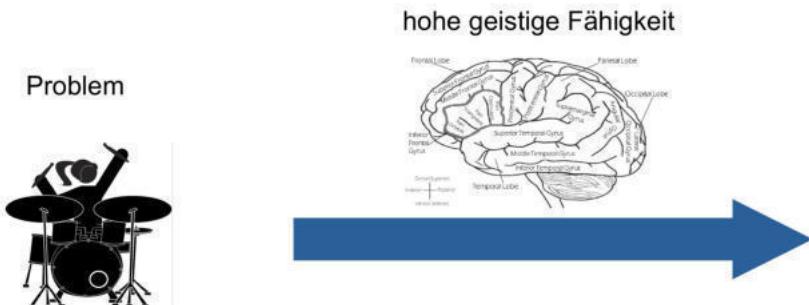


```
C:\windows\system32\cmd.exe
*****
Durchlauf 0
Playing hihat.
Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.
*****
Durchlauf 1
Playing hihat.
Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.
*****
Durchlauf 2
Playing hihat.
Playing hihat.
Playing hihat.
Playing snare.
*****
Gitarre
```

- Üben Sie den Umgang mit While und Do-While Schleifen bevor Sie den Groove implementieren.

Der Entwicklungsvorgang

- Es gibt leider keinen „Trick“ wie man diesen Prozess meistern kann.
- Es gibt kein generell gültiges Kochrezept.



Lösung

```
int i=0;
do
{
    i++;
    hihat.play();
} while (i<3);
```

Deshalb gilt:

Softwareentwicklung muss, genau wie eine Sportart, ständig Trainiert werden.

Ohne Training kein Erfolg!!!!!!!!!!!!!!

Am Anfang ist Softwareentwicklung sehr schwer. Je mehr Training, desto leichter wird es!

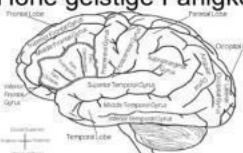
Forschungsgebiet in der Psychologie

- Softwareentwicklung ist Denkarbeit.
- Sie setzt hohe geistige Fähigkeiten beim Entwickler/bei der Entwicklerin voraus.
- Der Transfer eines „Problems“ aus der realen Welt in Codezeilen („Lösung“) ist ein Forschungsgebiet der Psychologie.
- Wie das Menschliche Gehirn diesen „Prozess“ meistert ist nicht geklärt.

Problem



Hohe geistige Fähigkeit



Lösung

```
int i=0;  
do  
{  
    i++;  
    hihat.play();  
} while (i<3);
```

2

Wie erzeugt unser Verstand aus Anforderungen einen Programmcode?



```
int i=0;  
  
do  
{  
    i++;  
    hihat.play();  
} while (i<3);
```

1

Generelles zur While Schleife

Die DO-WHILE - Anweisung nennt man auch **fußgesteuerte** bzw. **annehmende** Schleife, da die Bedingung am Fuß der Schleife steht, bzw. die Schleife mindestens einmal durchlaufen wird.



Aufgabe 5 a)

- Welchen Wert hat die Variable k nach dem Durchlauf der Schleife

```
byte k = 5;
while (k < 5)
{
    k--;
}
Console.WriteLine(k);
```

Aufgabe 5 b)

- Beschreiben Sie das Verhalten des Programms.

```
byte m = 5;
do
{
    m--;
} while (m < 5);
Console.WriteLine(m);
```

7

Aufgabe 6

- Stellen Sie das Programm als Struktogramm dar

```
byte m = 5;
do
{
    m--;
} while (m < 5);
Console.WriteLine(m);
```

8

Aufgabe 2, alle Zahlen von 0 bis 100 ausgeben

Aufgabe 3 a)

- Welchen Wert hat die Variable i nach dem Durchlauf der Schleife ?

```
int i=0;  
  
while (i < 5)  
{  
    i++;  
}  
Console.WriteLine(i);
```

Aufgabe 3 b)

- Welchen Wert hat die Variable t nach dem Durchlauf der Schleife ?

```
int t = 0;  
do  
{  
    t++;  
} while (t < 5);  
Console.WriteLine(i);
```

Do-While Schleife, Fußgesteuert

Anweisung(en) wenn Bedingung wahr.

```
do  
{  
    Anweisungen  
} while(Bedingung)
```

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

Bedingung → Wird letztendlich auf true/false (wahr/falsch) reduziert
d.h. auf den Daten Boolean reduziert.

Aufgabe 1, Endlosschleife, Code

Implementierne Sie eine Endlosschleife mit der do..while Anweisung.

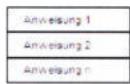
```
do {  
    Console.WriteLine("Gebe deinen Namen ein:");  
    string name = Console.ReadLine();  
}  
while(name != "");  
Console.WriteLine("Du heißt " + name);
```

Thema

Kontrollstrukturen IV Do-While-Anweisung

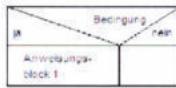
Struktogrammblöcke

Anweisungen

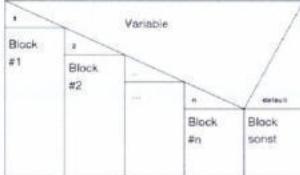


Kontrollstrukturen

Zweifachauswahl



Mehrfachauswahl

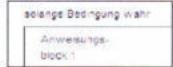


Schleifen

Zählergesteuerte Schleife



Kopfgesteuerte Schleife



Fußgesteuerte Schleife



Endlosschleife



Reflektion

- Die Bedingung einer While-Schleife wird während der Abarbeitung reduziert. Nennen Sie den Datentyp.
- Wie wird der zur While-Schleife zugehörige Block im Code dargestellt?
- Wie häufig wird eine While-Schleife durchlaufen?
- Worauf müssen Sie bei der Verwendung von Datentypen in der Bedingung der While-Schleife achten?

Frage:

Handelt sich es hier um eine Endlosschleife.

```
static void Main(string[] args)
{
    int i = 0;
    while (i >= 0)
    {
        i++;
    }
    Console.WriteLine("Fertig");
    Console.ReadKey();
}
```

Frage:

Wie oft wird der Schleifenrumpf einer kopfgesteuerten Schleife durchlaufen.

Nennen Sie die **obere** bzw. **untere** Grenze.

Generelles zur While Schleife

Die WHILE - Anweisung nennt man auch **kopfgesteuerte** bzw. **abweisende** Schleife, weil die Bedingung am Kopf der Schleife steht, bzw. die Schleife nicht durchlaufen werden muss (Durchgang wird abgewiesen)



Aufgabe 8

- Stellen Sie das Programm als Struktogramm dar

```
static void Main(string[] args)
{
    byte i = 0;
    while (i <= 10)
    {
        Console.WriteLine(i);
        i++;
    }

    Console.ReadKey();
}
```

Aufgabe 6

- Beschreiben Sie das Verhalten des Programms.

```
static void Main(string[] args)
{
    byte i = 0;
    while (i<=300)
    {
        Console.WriteLine(i);
        i++;
    }

    Console.ReadKey();
}
```

Aufgabe 7

- Beschreiben Sie das Verhalten des Programms.

```
static void Main(string[] args)
{
    byte i = 0;
    while (i <= 300) ;
    {
        Console.WriteLine(i);
        i++;
    }

    Console.ReadKey();
}
```

Aufgabe 5, alle Zahlen von 0 bis 100 ausgeben

- Elegante Lösung

Aufgabe 5, alle Zahlen von 0 bis 10 ausgeben

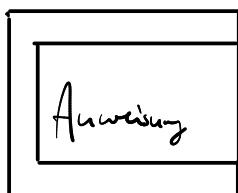
```
Console.WriteLine(1);
Console.WriteLine(2);
Console.WriteLine(3);
Console.WriteLine(4);
Console.WriteLine(5);
Console.WriteLine(6);
Console.WriteLine(7);
Console.WriteLine(8);
Console.WriteLine(9);
Console.WriteLine(10);
```

```
int a = 0;
while (a <= 10)
{
    Console.WriteLine(a);
    a++;
}
```

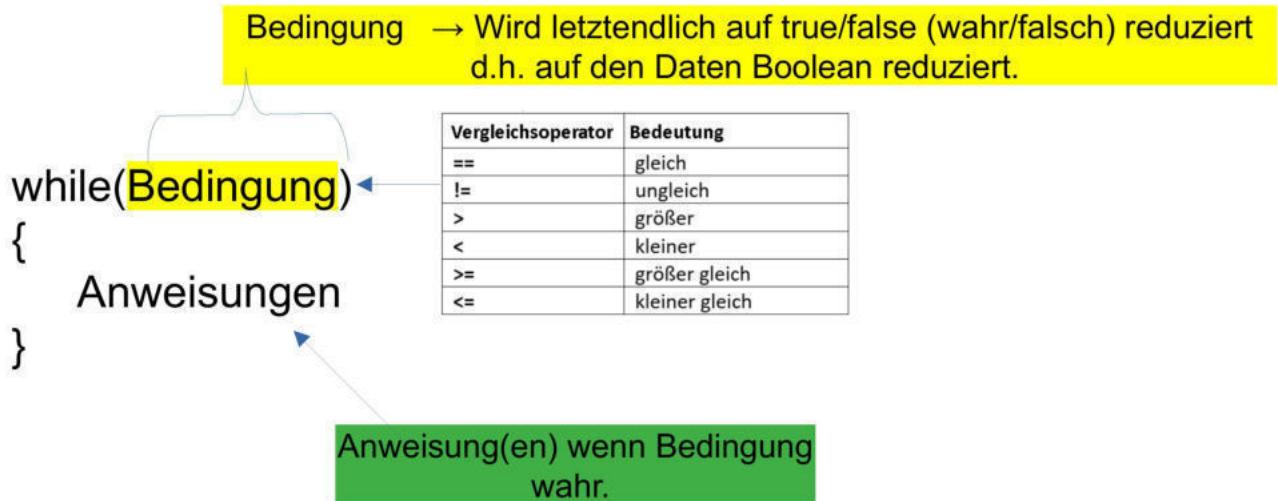
Aufgabe 4, Endlosschleife, Code

```
While (true) ....
{
    Anweisung
}
```

```
for ( ; ; )
{
    Anweisung
}
for ( ; true; )
{
    Anweisung
}
do {
    Anweisung
}
while (true);
```



While-Schleife (Kopfgesteuert)



Begrüßung

- Es soll unendlich oft das Wort „Hallo“ in der Konsole ausgegeben werden .
- Zeichnen Sie das Struktogramm.

A screenshot of a terminal window titled 'C:\Users\sasch\source...'. The window contains the word 'Hallo' repeated 15 times, demonstrating an infinite loop.

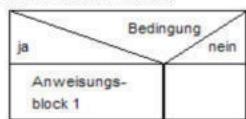
Struktogrammblöcke

Anweisungen

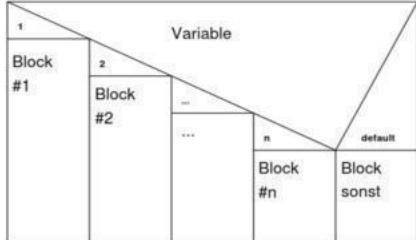


Kontrollstrukturen

Zweifachauswahl



Mehrfachauswahl



Schleifen

Zählergesteuerte Schleife



Kopfgesteuerte Schleife



Fußgesteuerte Schleife



Endlosschleife



Thema

Kontrollstrukturen III

While-Anweisung

Code / Struktogramm



Aufgabe

- Bestimmen Sie das Maximum zweier ganzer Zahlen.
- Zeichnen Sie das Struktogramm.
- Schreiben Sie den Code.

If-Anweisung, Verwendungsformen

- Welche Anweisungen werden ausgeführt? Kreuzen Sie an!

```
int eingabe = 1;  
int code = 2;
```

```
if (eingabe == code) Anweisung_a();   
else Anweisung_b(); 
```

```
if (eingabe == code) Anweisung_a();   
if (eingabe != code) Anweisung_b(); 
```

```
if (eingabe == code)  
    Anweisung_a();   
    Anweisung_b(); 
```

```
if (eingabe != code) Anweisung_c();   
Anweisung_d(); 
```

```
if (eingabe == code)
```

```
{  
    Anweisung_a();   
    Anweisung_b();   
}
```

```
if (eingabe != code)
```

```
{  
    Anweisung_c();   
    Anweisung_d();   
}
```

If-Anweisung, Verwendungsformen

- Welche Anweisungen werden ausgeführt? Kreuzen Sie an!

```
int eingabe = 1;  
int code = 1;
```

```
if (eingabe == code) Anweisung_a();   
else Anweisung_b(); 
```

```
if (eingabe == code) Anweisung_a();   
if (eingabe != code) Anweisung_b(); 
```

```
if (eingabe == code)  
    Anweisung_a();   
    Anweisung_b(); 
```

```
if (eingabe != code) Anweisung_c();   
Anweisung_d(); 
```

```
if (eingabe == code)
```

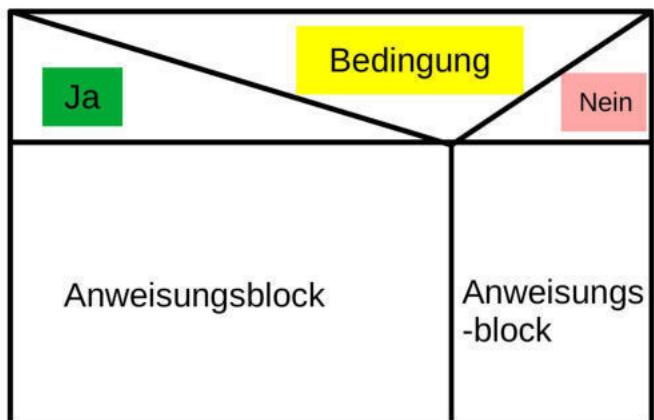
```
{  
    Anweisung_a();   
    Anweisung_b();   
}
```

```
if (eingabe != code)
```

```
{  
    Anweisung_c();   
    Anweisung_d();   
}
```

If-Anweisung, Struktogrammblock

```
Bedingung  
  
if (eingabe == code)  
{  
    Ja  
    Anweisungen a, b, c ...  
} else  
{  
    Nein  
    Anweisungen x, y, z ...  
}
```



If-Anweisung, die Bedingung

```
Bedingung → Wird letztendlich auf true/false (wahr/falsch) reduziert  
  
if (eingabe == code)  
{  
    Anweisungen a, b, c ... ← Nächste Anweisung wenn  
} else  
{  
    Anweisungen x, y, z ... ← Nächste Anweisung wenn  
}  
Bedingung wah.  
Bedingung falsch.
```

If-Anweisung

```
if (eingabe == code)
{
    Anweisungen a, b, c ...
} else
{
    Anweisungen x, y, z ...
}
```

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

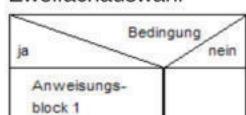
Struktogrammblöcke

Anweisungen

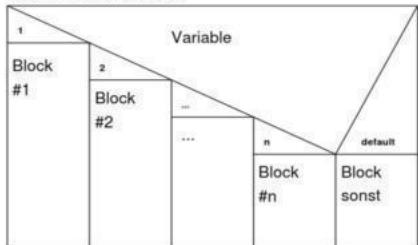


Kontrollstrukturen

Zweifachauswahl

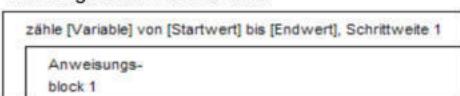


Mehrfachauswahl



Schleifen

Zählergesteuerte Schleife



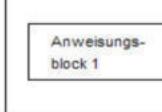
Kopfgesteuerte Schleife



Fußgesteuerte Schleife



Endlosschleife



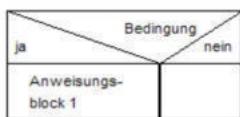
Struktogramm

Anweisungen

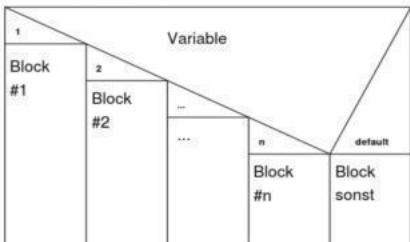
Anweisung 1
Anweisung 2
Anweisung n

Kontrollstrukturen

Zweifachauswahl

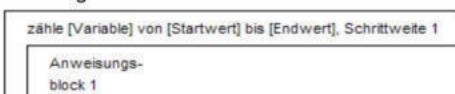


Mehrfachauswahl



Schleifen

Zählergesteuerte Schleife



Kopfgesteuerte Schleife



Fußgesteuerte Schleife



Endlosschleife



Klasse Convert, Funktionen

- Hier finden Sie alle Funktionen der Klasse Convert, nebst Beschreibung.

<https://docs.microsoft.com/de-de/dotnet/api/system.convert?view=net-5.0>

Funktion	Beschreibung
ToBoolean(Ausdruck)	Konvertieren des Ausdrucks nach Boolean.
ToByte(Ausdruck)	Konvertieren des Ausdrucks nach Byte.
ToChar(Ausdruck)	Konvertieren des Ausdrucks nach Char.
ToDecimal(Ausdruck)	Konvertieren des Ausdrucks nach Decimal.
ToDouble(Ausdruck)	Konvertieren des Ausdrucks nach Double.
ToInt32(Ausdruck)	Konvertieren des Ausdrucks nach Int32.

Ausdruck kann ein beliebiger Datentyp sein.

Im Zweifelsfall auf Microsoft Website nachschauen ob dieser unterstützt wird.

Besonderheiten Klasse Convert

- Es können Konverierungen durchgeführt werden, welche mit dem Typconvertierungsoperator nicht durchgeführt werden können.
- Convert überprüft Überläufe.
- Es können Strings in Zahlen konvertiert werden und Zahlen in Strings.

```
static void Main(string[] args)
{
    int value;
    string str;

    str = Console.ReadLine();
    value = Convert.ToInt32(str);
    //value = (int) str;           GEHT NICHT
    Console.WriteLine(str);
    Console.ReadKey();
```

Beispiel konvertieren String in Integer:

9

Explizite Konvertierung, Überlauf

Beispiel:

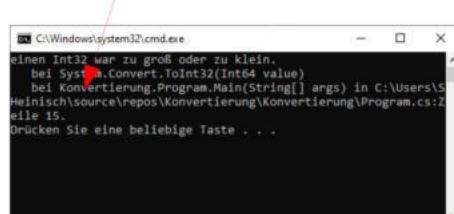
```
long wert1 = 40000000000;
int wert2;
wert2 = (int)wert1;
Console.WriteLine(wert2);
```

```
long wert1 = 40000000000;
int wert2;
wert2 = Convert.ToInt32(wert1);
Console.WriteLine(wert2);
```

falsch



Fehlermeldung



Eine Fehlermeldung ist sehr viel besser als ein falsches Ergebnis!

8

Explizite Konvertierung

- Die explizite Typkonvertierung erledigt der Programmierer.
- Es braucht zusätzlichen Programmcode.

Beispiel: Typkonvertierungsoperator prüft das Ergebnis nicht auf Überlauf.

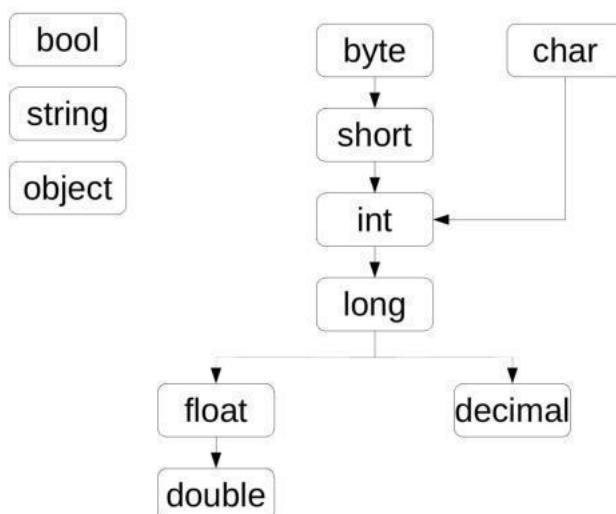
```
static void Main(string[] args)
{
    long wert1 = 11000;
    int wert2;
    wert2 = (int) wert1;
    Console.WriteLine(wert2);
    Console.ReadKey();
}
```

Klasse Convert prüft das Ergebnis auf Überlauf.

```
static void Main(string[] args)
{
    long wert1 = 11000;
    int wert2;
    wert2 = Convert.ToInt32(wert1);
    Console.WriteLine(wert2);
    Console.ReadKey();
}
```

Implizite Konvertierung

- Die implizite Typkonvertierung erledigt der C-Compiler selbstständig.
- Es braucht keinen zusätzlichen Programmcode.



Die Pfeilrichtung gibt die implizite Konvertierung an!

Beispiel implizite Konvertierung:

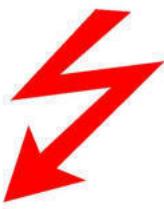
```
int wert1 = 11000;
long wert2;
wert2 = wert1;
```

Lösung

Steht bei der = Operation rechts ein anderer Datentyp als links, dann muss der rechte Datentyp in den linken umgewandelt (konvertiert) werden!

```
static void Main(string[] args)
{
    int wert1 = 11000;
    long wert2;
    wert2 = wert1;
    Console.WriteLine(wert2);
    Console.ReadKey();
}
```

```
static void Main(string[] args)
{
    long wert1 = 11000;
    int wert2;
    wert2 = wert1;
    Console.WriteLine(wert2);
    Console.ReadKey();
}
```



Der Wertebereich eines Integers ist kleiner als der eines Longs, siehe Seite 2!

Beispiel long → int

```
long wert1 = 11000;
int wert2;

wert2 = wert1;
```

- Geht das? Diskutieren Sie!

Beispiel int → long

```
int wert1 = 11000;  
long wert2;  
  
wert2 = wert1;
```

- Geht das? Diskutieren Sie!

3

Ganzzahlige Datentypen in C#

Typ	Speicherplatz	Wertebereich
byte	1 Byte	0 ... 255
sbyte	1 Byte	-128 ... 127
char	2 Byte	0 ... 65535
int	4 Byte	$-2^{31} \dots +2^{31}-1$
short	2 Byte	$-2^{15} \dots +2^{15}-1$
long	8 Byte	$-2^{63} \dots +2^{63}-1$

2

Thema

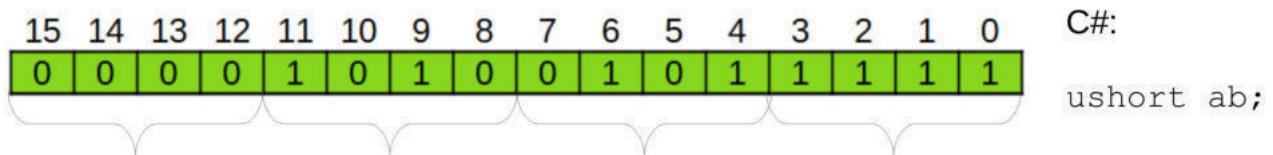
Elementare Typkonvertierung



Hausaufgabe

- 1) Wandeln Sie die Zahlen 4711_{dez} , 128_{dez} und 127_{dez} in das binäre Zahlensystem.
- 2) Wandeln Sie die Zahlen 4711_{dez} , 128_{dez} und 127_{dez} in das hexadezimale Zahlensystem.
- 3) Wandeln Sie die Zahl 10101010_{bin} in das dezimale Zahlensystem.
- 4) Wandeln Sie die Zahl 55_{hex} in das dezimale Zahlensystem.

Hexadezimale → Binär Umrechnung



Kochrezept

- 1) Binärzahl in Gruppen zu 4-Bit aufteilen.
- 2) Die 4-Bit Gruppen nach der Tabelle auf der vorherigen Seite Umwandeln.

Hexadezimale Darstellung von Zahlen

Hex.	Binär/Dual	Dezimal
0	0 0 0 0	0
1	0 0 0 1	1
2	0 0 1 0	2
3	0 0 1 1	3
4	0 1 0 0	4
5	0 1 0 1	5
6	0 1 1 0	6
7	0 1 1 1	7
8	1 0 0 0	8
9	1 0 0 1	9
A	1 0 1 0	10
B	1 0 1 1	11
C	1 1 0 0	12
D	1 1 0 1	13
E	1 1 1 0	14
F	1 1 1 1	15

Schreibweisen:

81₁₆
0x81 technische Informatik
81h technische Informatik
81H Intel
81_{HEX}
\$81 Motorola, MOS

- Das Hexadezimalsystem besteht aus 16 Zeichen, 0 bis F.
- Das Hexadezimalsystem wird sehr oft in der technischen Informatik verwendet.

Binäre Darstellung von Zahlen

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	byte
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	1	1	3
0	0	0	0	0	1	0	0	4
0	0	0	0	0	1	0	1	5
.								
0	1	1	1	1	1	1	0	126
0	1	1	1	1	1	1	1	127
1	0	0	0	0	0	0	0	128
1	0	0	0	0	0	0	1	129
.								
1	1	1	1	1	1	0	0	252
1	1	1	1	1	1	0	1	253
1	1	1	1	1	1	1	0	254
1	1	1	1	1	1	1	1	255

Beispiel einer Zahl in binärer Darstellung:

Zahlensystem

X= 010010101011111_{bin}

Aufgabe

- Berechnen Sie den Wertebereich für folgende Variable an.

C#:

ushort ab;

Aufgabe

- Berechnen Sie den Wertebereich für folgende Variable.

C#:

```
uint ab;
```

18

Aufgabe

- Skizzieren Sie den RAM-Speicherbedarf für folgende Variablen:

C#:

```
ubyte ax;  
ushort cf;  
uint ab;
```

Adresse
X+0
X+1
X+2
X+3
X+4
X+5
X+6
X+7
X+8
X+9

Bytes									

17

Aufgabe

- Skizzieren Sie den RAM-Speicherbedarf für folgende Variable:

C#:

```
uint ab;
```

Adresse	Bytes
X+0	
X+1	
X+2	
X+3	
X+4	
X+5	
X+6	
X+7	
X+8	
X+9	

16

Aufgabe

- Skizzieren Sie den RAM-Speicherbedarf für folgende Variable:

C#:

```
ushort ab;
```

Adresse	Bytes
X+0	
X+1	
X+2	
X+3	
X+4	
X+5	
X+6	
X+7	
X+8	
X+9	

15

Regel

- Die hier für die C# Datentypen byte und sbyte genannten Regeln gelten für alle anderen ganzzahligen Datentypen in gleicher Weise.
- Je mehr Bits ein Datentyp benötigt, desto mehr RAM-speicher wird für diesen belegt.
- Datentypen größer als ein Byte werden in aufeinanderfolgenden Bytes im Speicher abgelegt.
- Datentypen beginnen und enden an Bytегrenzen.

Wertebereich Datentyp „sbyte“

vz	2^6	2^5	2^4	2^3	2^2	2^1	2^0	sbyte
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	1	1	3
0	0	0	0	0	1	0	0	4
0	0	0	0	0	1	0	1	5
⋮								
0	1	1	1	1	1	1	0	126
0	1	1	1	1	1	1	1	127
1	0	0	0	0	0	0	0	-128
1	0	0	0	0	0	0	1	-127
⋮								
1	1	1	1	1	1	0	0	-4
1	1	1	1	1	1	0	1	-3
1	1	1	1	1	1	1	0	-2
1	1	1	1	1	1	1	1	-1

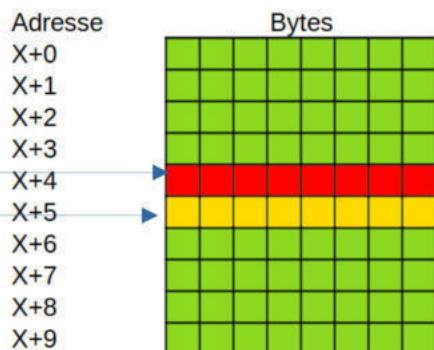
Datentyp „sbyte“

- Der Datentyp sbyte belegt ebenfalls ein Byte im Speicher.
- Das Bit Nummer Sieben hat jedoch die Funktion eines Vorzeichens.
- Der Wertebereich geht von -128 bis +127.

Beispiel:

C#:

```
byte ab;  
sbyte cd;
```



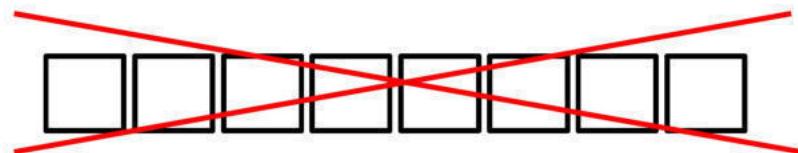
Wertebereich Datentyp „byte“

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	byte
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
0	0	0	0	0	0	1	1	3
0	0	0	0	0	1	0	0	4
0	0	0	0	0	1	0	1	5
⋮								
0	1	1	1	1	1	1	0	126
0	1	1	1	1	1	1	1	127
1	0	0	0	0	0	0	0	128
1	0	0	0	0	0	0	1	129
⋮								
1	1	1	1	1	1	0	0	252
1	1	1	1	1	1	0	1	253
1	1	1	1	1	1	1	0	254
1	1	1	1	1	1	1	1	255

Byte

Wertebereich Datentyp „byte“

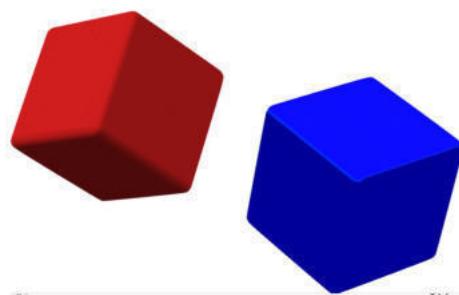
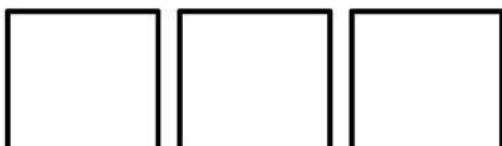
- Wie viele „Zustände“ können 8 Bits annehmen?
- Berechnen Sie.



- Es können ____ angenommen werden. Der Zahlenbereich geht von ____ bis ____.

Wertebereich Datentyp „byte“

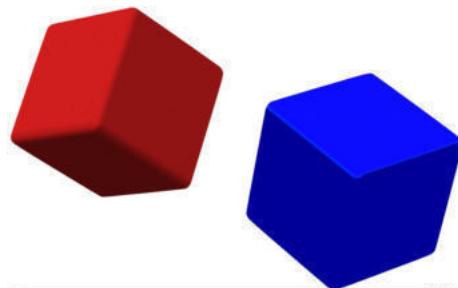
- Wie viele „Zustände“ können vier Bits annehmen?
- Roter Würfel = 1, blauer Würfel = 0



- Es können ____ angenommen werden. Der Zahlenbereich geht von ____ bis ____.
- Formel:
 $(n = \text{Anzahl Bits}, wt = \text{Wertebereich})$

Wertebereich Datentyp „byte“

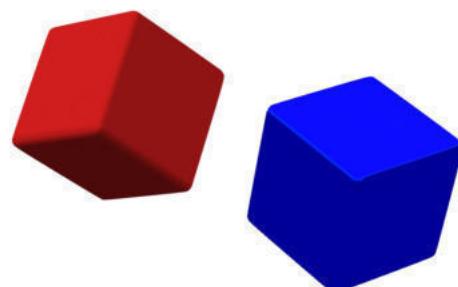
- Wie viele „Zustände“ können zwei Bits annehmen?
- Roter Würfel = 1, blauer Würfel = 0



- Es können ____ angenommen werden. Der Zahlenbereich geht von ____ bis ____.

Wertebereich Datentyp „byte“

- Wie viele „Zustände“ kann ein Bit annehmen?
- Roter Würfel = 1, blauer Würfel = 0



- Es können ____ angenommen werden. Der Zahlenbereich geht von ____ bis ____.

Speicherbedarf Datentyp „byte“

- Der Datentyp Byte belegt ein Byte im Ram-Speicher.

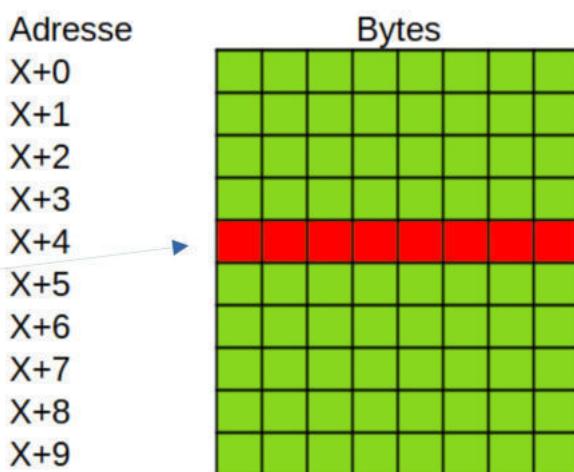
- Der Ram Speicher ist Byte-weise organisiert. Adresse

- Jedes Byte hat eine Adresse.

Beispiel:

C#:

byte ab;



Der Datentyp „byte“

- 1 Byte = 8 Bit.

Beispiel:

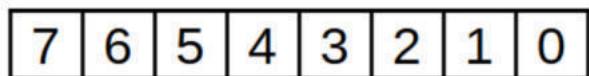
7 6 5 4 3 2 1 0 ← Bit Number

$$78_{\text{dez}} = \begin{array}{cccccccc} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ \downarrow & & & & & & & \uparrow \\ \text{MSB} & & & & & & & \text{LSB} \end{array}$$

LSB = Least Significant Bit (niederwertigstes Bit)
MSB = Most Significant Bit (höchstwertigstes Bit)

Der Datentyp „byte“

- 1 Byte = 8 Bit.
- Byte 8 Bit = 1 Byte



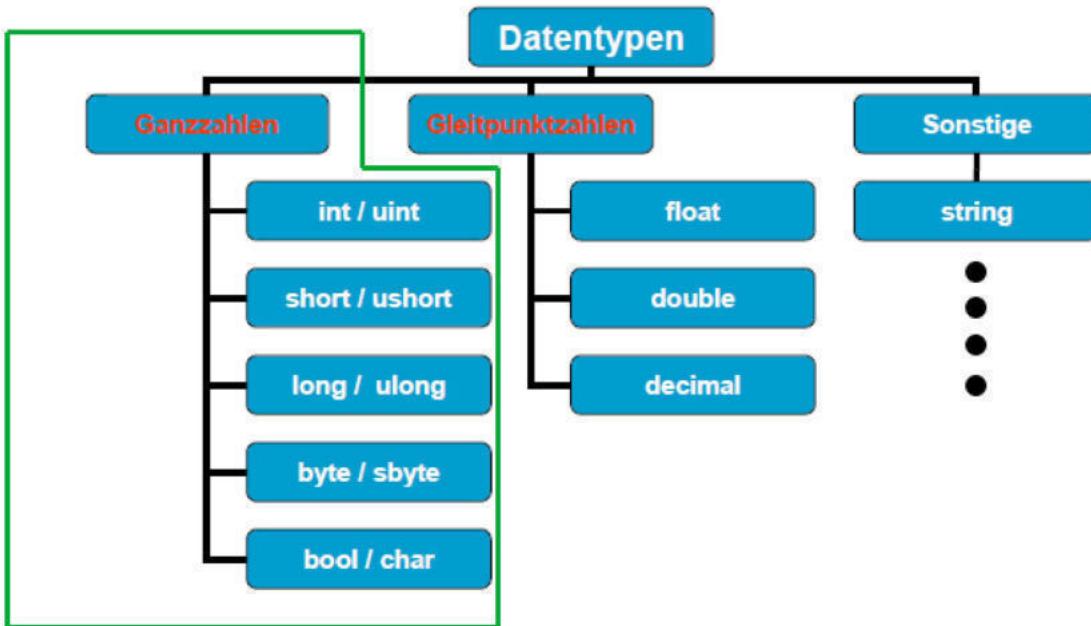
- Wertebereich:

00_{dez} bis 255_{dez} = 0_{hex} bis FF_{hex} = 00000000_{bin} bis 11111111_{bin}

Ganzzahlige Datentypen in C#

Typ	Speicherplatz	Wertebereich
byte	1 Byte	0 ... 255
sbyte	1 Byte	-128 ... 127
char	2 Byte	0 ... 65535
int	4 Byte	$-2^{31} \dots +2^{31}-1$
short	2 Byte	$-2^{15} \dots +2^{15}-1$
long	8 Byte	$-2^{63} \dots +2^{63}-1$

Elementare Datentypen



Logical vs. physical address of the stack

- Calculating the physical address for the stack, the same principle is applied as was used for the code and data segments. Physical address depends on the value of stack segment (SS) register and the stack pointer (SP).

Ex: If SS=3500H and SP:FFFEH

- a) Calculate the physical address: $35000+FFFE = 44FFE$
- b) Calculate the lower range of the stack: $35000+0000 = 35000$
- c) Calculate the upper range of the stack segment: $35000+FFFF = 44FFF$
- d) Show the logical address of the stack: $3500:FFFE$

THE FLAG REGISTER (FR) AND BIT FIELDS

- The flag register is a 16-bit register sometimes referred as the ***status*** register. Although the register is 16-bit. Not all the bits are used.
- ***Conditional flags:*** 6 of the flags are called the conditional flags, meaning that they indicate some condition that resulted after an instruction was executed. These 6 are: CF, PF, AF, ZF, SF, and OF.
- The 16 bits of the flag registers:

R	R	R	R	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF
R= reserved	U= undefined	OF= overflow flag	DF= direction flag	IF= interrupt flag	TF= trap flag	SF= sign flag	ZF= zero flag	AF= auxiliary carry flag	PF= parity flag	CF= carry flag					

CF, the Carry Flag: This flag is set whenever there is a carry out, either from d7 after an 8-bit operation, or from d15 after a 16-bit data operation.

PF, the Parity Flag: After certain operations, the parity of the result's low-order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise, it is cleared.

AF, the Auxiliary Carry Flag: If there is a carry from d3 to d4 of an operation this bit is set to 1, otherwise cleared (set to 0).

ZF, the Zero Flag: The ZF is set to 1 if the result of the arithmetic or logical operation is zero, otherwise, it is cleared (set to 0).

SF, the Sign Flag: MSB is used as the sign bit of the binary representation of the signed numbers. After arithmetic or logical operations the MSB is copied into SF to indicate the sign of the result.

TF, the Trap Flag: When this flag is set it allows the program to single step, meaning to execute one instruction at a time. Used for debugging purposes.

IF, Interrupt Enable Flag: This bit is set or cleared to enable or disable only the external interrupt requests.

DF, the Direction Flag: This bit is used to control the direction of the string operations.

OF, the Overflow Flag: This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.

STACK: PUSHING AND POPPING OPERATIONS

What is a stack, and why is it needed?

The stack is a section of read/write memory (RAM) used by the CPU to store information temporarily. CPU needs this storage area since there are *only limited number* of registers.

How stacks are accessed

- SS (stack segment) and SP (stack pointer) must be loaded to access stack in the memory.
- Every register in the CPU (except segment registers and SP) can be stored in the stack and loaded from the stack.

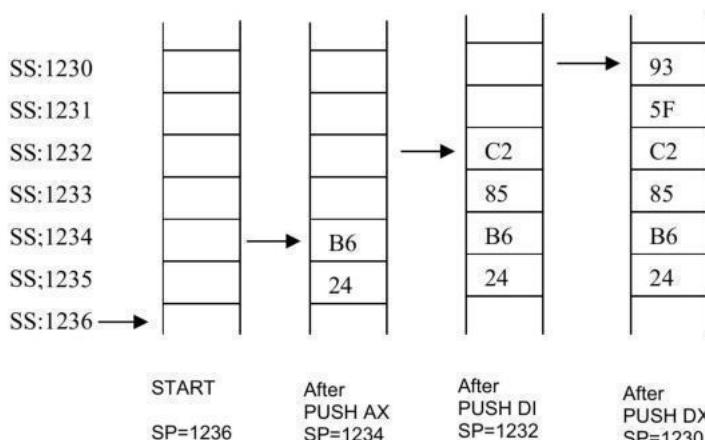
Pushing onto the stack

- Storing the CPU register in the stack is called a *push*.

Ex: SP=1236, AX=24B6, DI=85C2, and DX=5F93, show the contents of the stack as each instruction is executed.

PUSH AX
PUSH DI
PUSH DX

Solution:



- Note that in 80x86 the lower byte of the register is stored to the lower address.

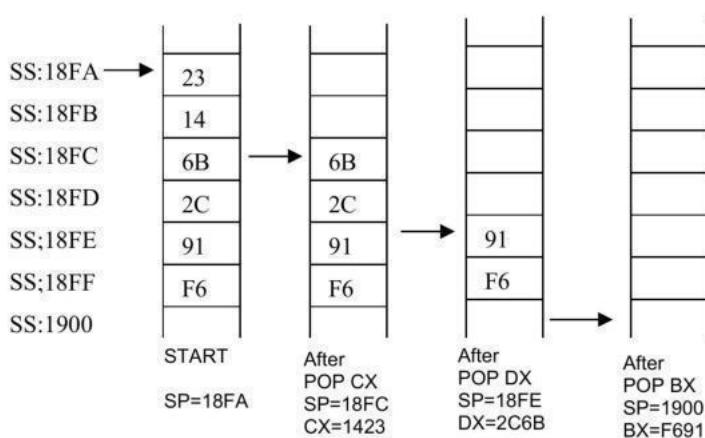
Popping the stack

- Loading the contents of the stack into the CPU register is called a *pop*.

Ex: assume that the stack is shown below, and SP=18FA, show the contents of the stack and registers as each of the following instructions is executed.

POP CX
POP DX
POP BX

Solution:



Microprocessors Memory Map

Outline of the Lecture

- **Memory Map of the IBM PC**
- **Pushing and Popping Operations (Stack)**
- **Flag Registers and bit fields**

MEMORY MAP OF THE IBM PC

- The 20-bit address of the 8086/8088 allows 1M byte of (1024 K bytes) memory space with the address range 00000-FFFFF.
- The allocation of the memory is called a ***memory map***.

RAM 640K	00000H
	9FFFFH
Video Display RAM 128K	A0000H
	BFFFFH
ROM 256K	C0000H
	FFFFFH

RAM: Memory locations from 00000H to 9FFFFH (640K) are set aside for RAM. In an IBM PC the DOS operating system first allocates the available RAM on the PC for its own use and let the rest be used for applications such as word processors.

The amount of memory used by DOS varies among its various versions. That is why we do not assign any values for the CS, DS, SS, and ES. Memory management functions within DOS handle this for the operating system.

Video RAM: Memory locations from A0000H to BFFFFH (128K) are set aside for video. This amount varies depending on the video board installed on the PC.

ROM: Memory locations from C0000H to FFFFFH (256K) are set aside for ROM. First 64 K bytes is used by BIOS (Basic Input/Output System) ROM. Some of the remaining space is used for adapter cards.

Function of BIOS ROM:

- CPU can only execute programs that are stored in memory, there must be some permanent (nonvolatile) memory to hold the programs telling the CPU what to do when the power is turned on.
- BIOS contains programs to test RAM and other components connected to the CPU.

Übung 2 zu Variablen

Welcher Datentyp ist für folgende Initialisierungen geeignet. Ergänzen Sie entsprechend und überprüfen Sie die am Rechner.

```
zahl = 3;  
x = 3.87f;  
y = 9.8235643;  
buchstabe = 'D';  
auswahl = '8';  
einwohner = 65000000000000;  
ergebnis = 0.9e200;  
hallo = "Hallo";  
bestanden = true;  
genauigkeit = 0.8734839243764372647m;
```

Übung 1 zu Variablen

Testen Sie, welche der folgenden Variablennamen zulässig sind! Suchen Sie eine Begründung.

Mehrwertsteuer,
3rad,
auto?,
vierrad,
int,
länge,
extern,
do it

Schlüsselworte in C#

Achtung:
nicht als
Variablenamen
verwenden!

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

Welche Einschränkungen gelten für Bezeichner?

Erstellen Sie eine neue Konsolenanwendung und
testen Sie folgende Variablendeklarationen!

Weshalb sind diese Bezeichner nicht zulässig?

int meine neue Variable = 7;
char ja! = '1';
double do = 7;
int zähler = 0;

Deklaration

Bezeichner werden innerhalb eines Programms durch **Deklaration** bekannt gemacht:

```
int zaehler;  
char ch;  
double x,y,z;
```

Syntax: Datentyp name1 [, name2 ...];

Anmerkung: Die **Syntax** definiert die äußen Formgesetze einer Programmiersprache (ähnlich den grammatischen Regeln einer natürlichen Sprache).

Initialisierung

Bei einer Definition ist es gleichzeitig möglich einen Bezeichner zu initialisieren, d.h. ihm einen Anfangswert zuzuweisen.

```
int zaehler = 1;  
char ch1 = 'c', ch2 = (char) 65;  
double x = -1.56, y = 3.0e10, z = 7;
```

Definition

Die **Definition** ist ein Sonderfall der **Deklaration**.

Bei Variablen spricht man von Definition, wenn der Übersetzer Code erzeugt, der Speicherplatz für diese Variable reserviert.

ASCII, American Standard Code for Information Interchange

=Codierungsschema, das Zeichen, Zahlen, Interpunktionszeichen und bestimmten Sonderzeichen jeweils numerische Werte zuordnet

Unicode-Zeichen von 0 bis 127: C0 Controls and Basic Latin

alle 128 Zeichen (grauer Hintergrund) und in numerischer Auszeichnung (gelber Hintergrund).

�	�	□		□		□		□		□		□		□	
□																
□															
□															
	 	!	!	"											
(())	*											
0	0	1	1	2											
8	8	9	9	:											
@	@	A	A	B											
H	H	I	I	J											
P	P	Q	Q	R											
X	X	Y	Y	Z												
	`	a	a	b												
h	h	i	i	j	
										
p	p	q	q	r											
x	x	y	y	z											

Unicodezeichensatz: C1 Controls and Latin-1 Supplement - Windows Internet Explorer

Unicode-Zeichen von 128 bis 255: C1 Controls and Latin-1 Supplement

alle 128 Zeichen (grauer Hintergrund) und in numerischer Auszeichnung (gelber Hintergrund).

€ €	□ 	„ ‚	ƒ ƒ	„ „	… …	† †	‡ ‡
“ ˆ	% ‰	Š Š	„ ‹	Œ Œ	□ 	Ž Ž	□ 
□ 	“ ‘	„ ’	“ “	“ ”	• •	– –	— —
~ ˜	™ ™	đ š	› ›	œ œ	□ 	ž ž	Ÿ Ÿ
฿ 	฿ ¡	ດ ¢	₭ £	₱ ¤	₩ ¥	₭ ¦	₪ §
– ¨	⌚ ©	⌚ ª	⌚ «	⌚ ¬	⌚ ­	⌚ ®	⌚ ¯
⌚ °	⌚ ±	⌚ ²	⌚ ³	⌚ ´	⌚ µ	⌚ ¶	⌚ ·
⌚ ¸	⌚ ¹	⌚ º	⌚ »	⌚ ¼	⌚ ½	⌚ ¾	⌚ ¿
À À	Á Á	Â Â	Ã Ã	À Ä	Á Å	Ã Æ	Ҫ Ç
È È	É É	Ê Ê	Ë Ë	Ì Ì	Í Í	Ï Î	Ӯ Ï
Ծ Ð	Ը Ñ	Ծ Ò	Ը Ó	Ծ Ô	Ը Õ	Ը Ö	Ӯ ×
Ծ Ø	Ը Ù	Ը Ú	Ը Û	Ը Ü	Ը Ý	Ը Þ	Ը ß
Ծ à	Ծ á	Ծ â	Ծ ã	Ծ ä	Ծ å	Ծ æ	Ծ ç
Ծ è	Ծ é	Ծ ê	Ծ ë	Ծ ì	Ծ í	Ծ î	Ծ ï
Ծ ð	Ծ ñ	Ծ ò	Ծ ó	Ծ ô	Ծ õ	Ծ ö	Ծ ÷
Ծ ø	Ծ ù	Ծ ú	Ծ û	Ծ ü	Ծ ý	Ծ þ	Ծ ÿ

Fertig

Internet

100%

Unicodestandard-Übersicht

Übersicht über den Standard

Der Unicode-Standard mit 65536 möglichen Zeichen ist der aktuell umfassendste Zeichensatzstandard. Nachfolgende Tabelle enthält die Aufzählung der Sprachzeichensätze und ihrer Adressierung im Unicodestandard in hexadezimaler und numerischer Form.

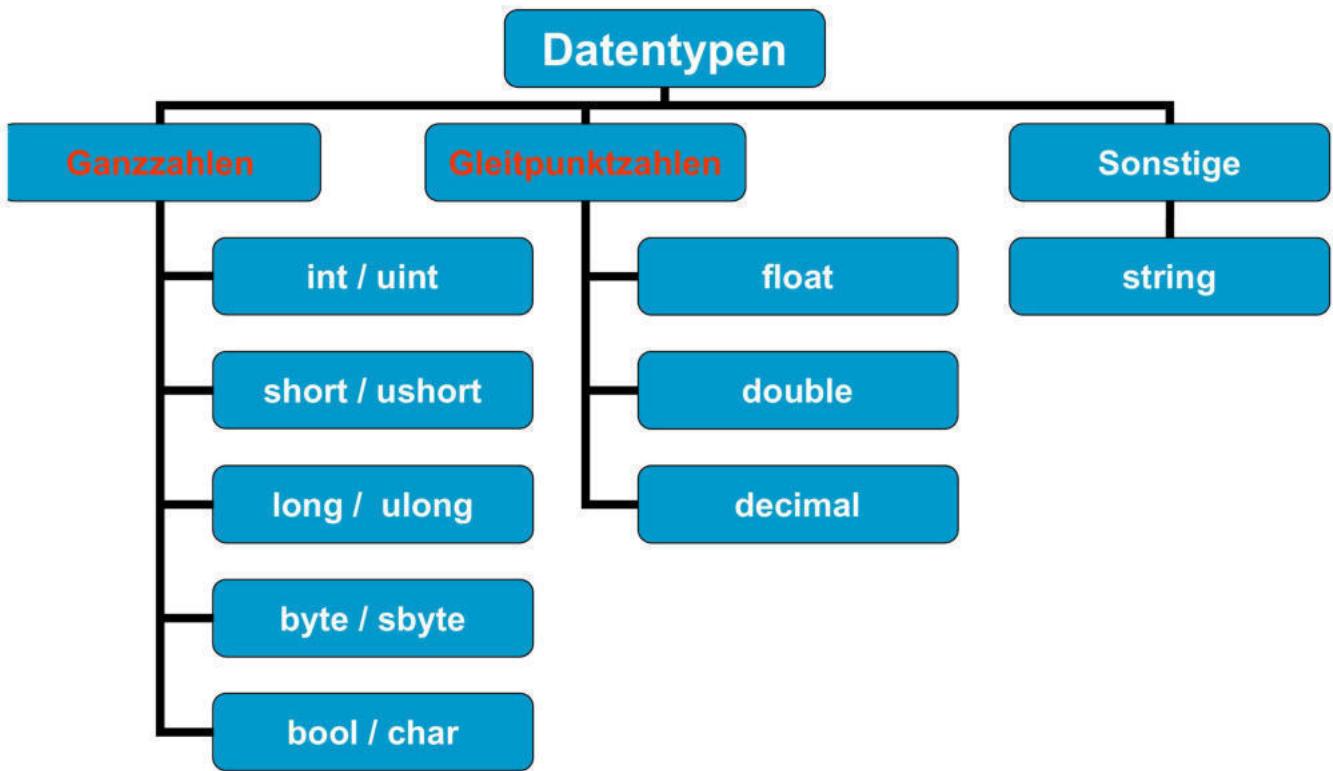
Bisher gibt es keine True-Type-Schrift (ttf), die den gesamten Unicode-Zeichensatz enthält. An nächsten kommt True-Type-Datei **Arial Unicode MS** von The Monotype Corporation (z.B. in MS Office 2000 enthalten - OFFICE1.CAB) mit einer Dateigröße von 23.566 KByte, welche allerdings noch reichlich Lücken aufweist.

Bitte beachten: Das Anzeigen der Zeichen funktioniert nur, wenn auch die passende unicodefähige Schriftdatei auf dem Computer-System installiert ist. Je nach Zeichenzahl kann die Anzeige entsprechend lange dauern, da die Seite mit den Unicode-Zeichen per JavaScript erstellt wird.

Sprache - Zeichensatzname	Unicode-hexcode	Unicode-numerisch	Zeichen
C0 Controls and Basic Latin	U+0000 - U+007F	(0-127)	anzeigen
C1 Controls and Latin-1 Supplement	U+0080 - U+00FF	(128-255)	anzeigen
Latin Extended-A	U+0100 - U+017F	(256-383)	anzeigen
Latin Extended-B	U+0180 - U+024F	(384-591)	anzeigen
IPA Extensions	U+0250 - U+02AF	(592-687)	anzeigen
Spacing Modifier Letters	U+02B0 - U+02FF	(688-767)	anzeigen
Combining Diacritical Marks	U+0300 - U+036F	(768-879)	anzeigen
Greek	U+0370 - U+03FF	(880-1023)	anzeigen
Cyrillic	U+0400 - U+04FF	(1024-1279)	anzeigen

www.free-solutions.de/w3/zeichensatz_unicodestandard.html

Elementare Datentypen



Speicherplatz der Datentypen

Typ	Speicherplatz	Wertebereich
byte	1 Byte	0 ... 255
sbyte	1 Byte	-128 ... 127
char	2 Byte	0 ... 65535
int	4 Byte	$-2^{31} \dots +2^{31}-1$
short	2 Byte	$-2^{15} \dots +2^{15}-1$
long	8 Byte	$-2^{63} \dots +2^{63}-1$
float	4 Byte	$\pm 1.5 \times 10^{-45}$ bis $\pm 3,4 \times 10^{38}$
double	8 Byte	$\pm 5.0 \times 10^{-324}$ bis $\pm 1,7 \times 10^{308}$
decimal	16 Byte	$\pm 1.0 \times 10^{-28}$ bis $\pm 7,9 \times 10^{28}$

Tabelle der Elementaren Datentypen

Datentyp	Anzahl Bytes	Kleinster Wert	Größter Wert
----------	--------------	----------------	--------------

Your Turn!

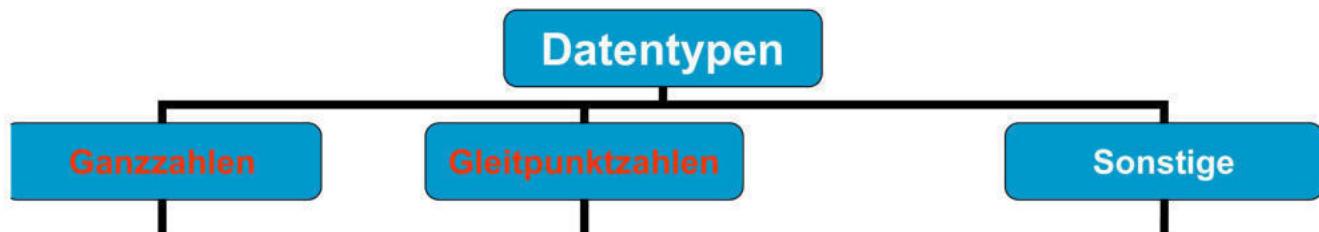
Erstellen Sie ein Konsolenprojekt und geben Sie Datentyp, Anzahl Bytes, kleinster und größter Wert für alle Datentypen als Tabelle aus. Die Konsolenausgabe ausdrucken und hier einkleben.

Programmiertipp:

```
Console.WriteLine("int\t{0}\t{1}\t{2}",  
    sizeof(int), int.MinValue, int.MaxValue);
```



Elementare Datentypen



Deklaration

Bezeichner werden innerhalb eines Programms durch **Deklaration** bekannt gemacht:

```
int zaehler;  
char ch;  
double x,y,z;
```

Syntax: Datentyp name1 [, name2 ...];

Anmerkung: Die **Syntax** definiert die äußeren Formgesetze einer Programmiersprache (ähnlich den grammatischen Regeln einer natürlichen Sprache).

Anton Schimmerlos

„Variablen“

- Deklaration
- Definition
- Initialisierung



Reflektion

- Wie lange wird ein Software Lebenszyklus ausgeführt?
- Aus welchem Grund ist eine Wartung/Pflege von Software notwendig?
- Kann man auf die Einführung verzichten?
- Woran kann die Softwareentwicklung scheitern?

Gefahren



- Viele Softwareprojekte scheitern.
 - Wesentliche Produkteigenschaften fehlen zum Zeitpunkt der Markteinführung.
 - Es wurde das falsche Produkt entwickelt.
 - Das Design ist so komplex, dass die Software „nie“ vollendet wird.
 - Der Programmcode ist so komplex, dass die Software „nie“ vollendet wird.
 - Die Software hat viele Fehler.

<https://www.silicon.de/39200412/68-prozent-aller-it-projekte-scheitern>

Probleme während der Entwicklung



Verkäufer*innen

- Versprechen dem Kunden unrealistische Produkteigenschaften.
 - Kunden werden enttäuscht, wandern zur Konkurrenz.
- Abhilfe: Produktschulung durch Entwickler.

Probleme während der Entwicklung



Entwickler*innen

- Kündigen oder werden versetzt.
 - Wichtiges Wissen wandert ab.
 - Entwicklung wird verzögert, Termine werden nicht eingehalten.

Abhilfe: Versetzungsstopp während der Entwicklung. Gute Bezahlung, gutes Arbeitsklima. Mitspracherecht bei Entscheidungen. Gute Arbeitsumgebung.

Probleme während der Entwicklung



Entwickler*innen

- Verstehen Auftraggeber nicht.

→ Es wird das falsche Produkt entwickelt. Kosten trägt der Auftragnehmer.

Abhilfe: Geeignete Vorgehensmodelle anwenden, keinen Zeitdruck, Schulungen, frühe Prototypen dem Auftraggeber vorstellen, Angestellte des Auftraggebers befragen.

- Wollen am liebsten nur programmieren. Modelle, Design, Dokumentation usw. sind langweilig.

→ Softwarequalität wird schlecht, Wartung wird aufwändig.

→ Software wird nie fertig.

Abhilfe: Vorgehensmodelle und Prozesse als verbindlich vorgeben. Anwendung durch eine Qualitätsabteilung überprüfen lassen.

Probleme während der Entwicklung



Auftraggeber*innen

- Erweitern ihre Wünsche während der Implementierung.

→ Entwicklungszeiten verlängern sich zum Teil drastisch.

→ Design muss teilweise bei kleinen Änderungen komplett überarbeitet werden.

→ Bisher entwickelte Software unbrauchbar.

Abhilfe: Vertraglich festhalten, dass Kosten für Erweiterungen bezahlt werden müssen, anwenden von Agilen Vorgehensmodellen.

Probleme während der Entwicklung



Auftraggeber*innen

- Ändern ihre Wünsche während der Implementierung.
 - Entwicklungszeiten verlängern sich zum Teil drastisch.
 - Design muss teilweise bei kleinen Änderungen komplett überarbeitet werden.
 - Bisher entwickelte Software unbrauchbar.
- Abhilfe: Vertraglich festhalten, dass Kosten für Änderungen bezahlt werden müssen, anwenden von Agilen Vorgehensmodellen.

Probleme während der Entwicklung



Auftraggeber*innen

- Wollen einen schnelle Lösung.
 - Gute Software kann nicht unter Zeitdruck entstehen. Es kommt zu vielen Fehlern während der Entwicklung. Die Kosten der Fehler will der Auftraggeber nicht bezahlen.

Abhilfe: Entwickler müssen während den Verhandlungen den Vertrieb unterstützen. Ggf. eine Risikoanalyse anfertigen und die Gefahren insbesondere die Kosten ermitteln zu denen die Fehler führen könnten.

Probleme während der Entwicklung



Auftraggeber*innen

- Können ihr Problem nicht so beschreiben, dass Entwickler dieses versteht.

→ Es wird das falsche Produkt entwickelt. Auftraggeber will dafür nicht zahlen.

Abhilfe: Vertrag mit Bestätigung der Anforderungen, früh Prototypen zeigen, stetig im Kontakt bleiben. Agile Vorgehensmodelle, Angestellte der Auftraggebers befragen.

- Können ihr Problem nicht formal beschreiben, z.B. in Form von Anforderungen.

→ Es wird das falsche Produkt entwickelt. Auftraggeber will dafür nicht zahlen.

Abhilfe: Vertrag mit Bestätigung der Anforderungen, früh Prototypen zeigen, stetig im Kontakt bleiben, Agile Vorgehensmodelle anwenden.

Dokumentation Gründe

- Schneller Einarbeitung von neuen Entwicklern.
- Teilweise treten Fehler erst nach Jahren in der Anwendung auf. Oftmals existiert das Entwickler*innen Team nicht mehr.

Beispiel:

- Jahr 2000 Problem, Software kann Jahreszahlen > 1999 nicht verarbeiten.
 - Banken verwenden bis heute Großrechner aus den 60er Jahren.
 - Software wurde in Cobol geschrieben.
 - Entwickler*innen sind schon seit vielen Jahren in Rente.
 - Entwickler*innen wurden aus der Rente zurück geholt. Diese mussten sich wieder einarbeiten.

Dokumentation

- Eine Software sollte Grundsätzlich dokumentiert werden.
- Diese Dokumentation besteht aus:
 - Analyse
 - Design
 - Teilen der Implementierung
 - Testprotokollen
- In vielen Fällen ist es notwendig eine Anleitung für den Anwender sowie Schulungsunterlagen zu erstellen.



Änderungen

-
-

Problem:

Der Zähler soll vom Hausmeister auf 0 gesetzt werden können.

Bugtracking:

Prioritäten setzen:



Von Sven Krohlas -
<http://krohlas.de> - Eigenes
Werk, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=307046>

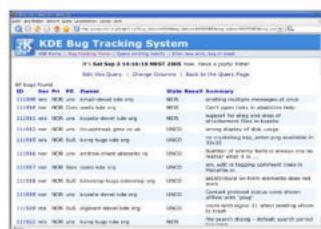


Pflege/Wartung

Problem:

Der Kunde hat einen Fehler gemeldet.

Bugtracking: Bugzilla:



Von Sven Krohlas -
<http://krohlas.de> - Eigenes
Werk, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=207046>



Entwicklungsprozess

9

Einführung

Problem:

Parkhausbetreiber möchte wissen wie viele Kraftfahrzeuge pro Tag in ein Parkhaus hineinfahren

Lösung:



Entwicklungsprozess

8

Der Test

•

Problem:

Parkhausbetreiber möchte wissen wie viele Kraftfahrzeuge pro Tag in ein Parkhaus hinein fahren.

Test:



Implementierung (Umsetzung)

•

Problem:

Parkhausbetreiber möchte wissen wie viele Kraftfahrzeuge pro Tag in ein Parkhaus hinein fahren.

Programmcode:

C#:

```
static void Main(string[] args)
{
    int Zähler;
    while (true)
    {
        Zähler = 0;
        while (false == dayover())
        {
            if (true == readLightBarrier())
            {
                Zähler = Zähler + 1;
            }
            else
            {
            }
            Console.Clear();
            Console.WriteLine(Zähler);
        }
    }
}
```

Der Entwurf (Das Design)

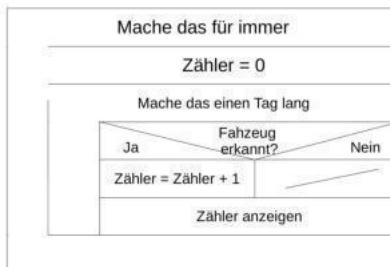
•

Problem:

Parkhausbetreiber möchte wissen wie viele Kraftfahrzeuge pro Tag in ein Parkhaus hinein fahren.

Entwurf:

Struktogramm:



Entwicklungsprozess

5

Die Analyse

Problem:

Parkhausbetreiber möchte wissen wie viele Kraftfahrzeuge pro Tag in ein Parkhaus hinein fahren.

Requirement 100:

Wenn ein Fahrzeug einfährt soll dieses gezählt werden.

Requirement 200:

Die gezählten Fahrzeuge sollen angezeigt werden.

Requirement 300:

Es wird jeden Tag neu mit Zählen begonnen.



Entwicklungsprozess

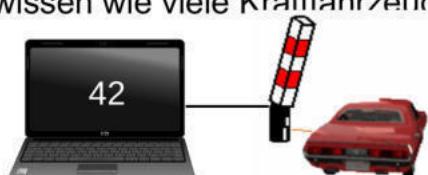
4

Die Problemstellung

-
-
-

Problem:

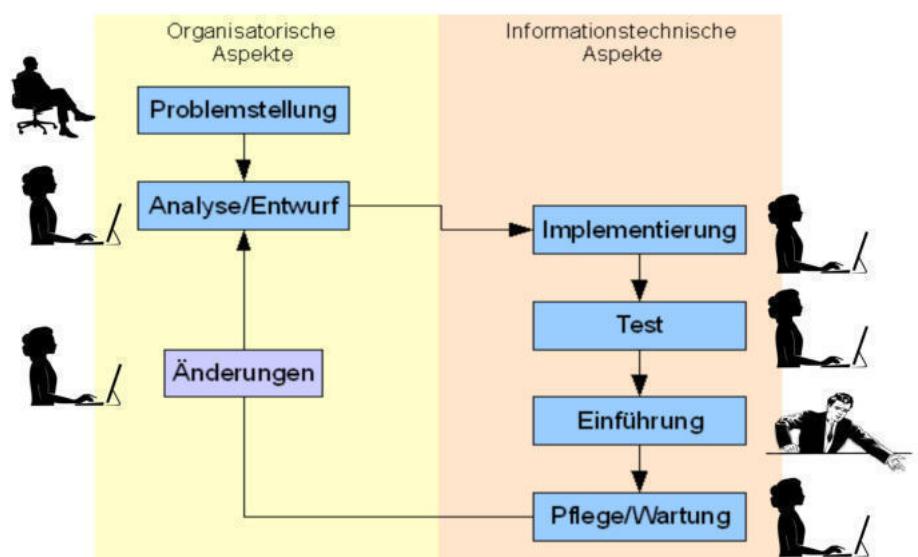
Parkhausbetreiber möchte wissen wie viele Kraftfahrzeuge pro Tag in ein Parkhaus hinein fahren.



Der Software-Lebenszyklus

Aus der Wikipedia:

-  Auftraggeber*innen
-  Entwickler*innen
-  Verkäufer*innen



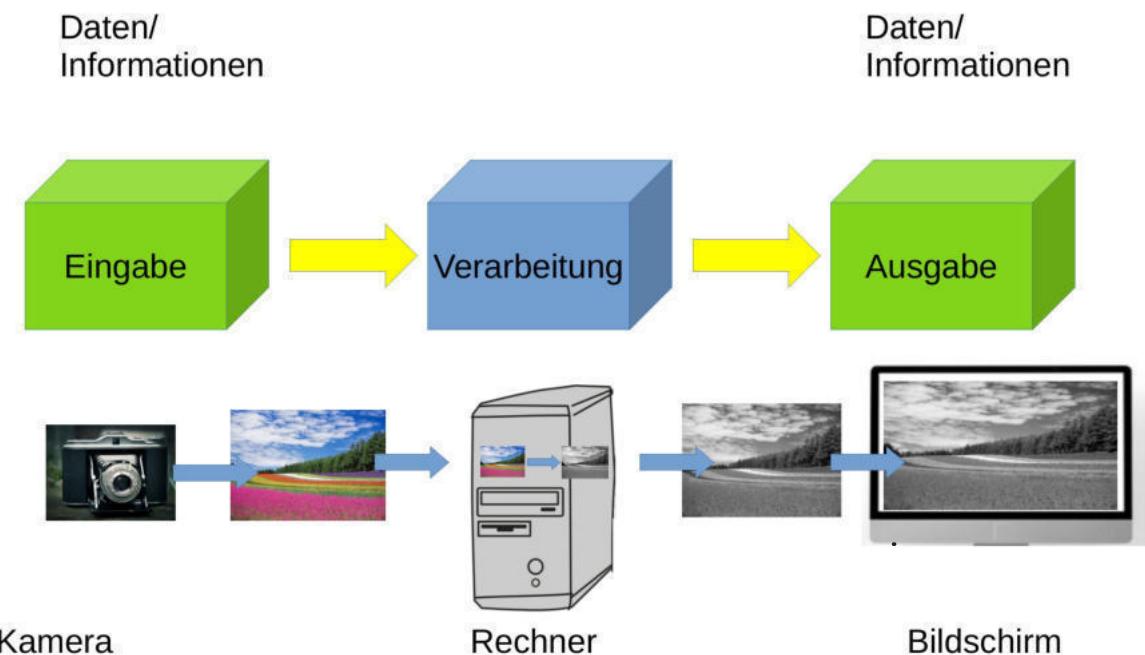
Thema



Reflektion

- Welchen Aufbau haben Daten?
- Kann ein Datum (Singular von Daten) aus einem Zeichen bestehen?
- Kann ein Zeichen aus einem Bit bestehen?
- Kann ein Datum aus einem Bit bestehen?
- Ist eine Datenverarbeitungsanlage ohne Eingabe denkbar?

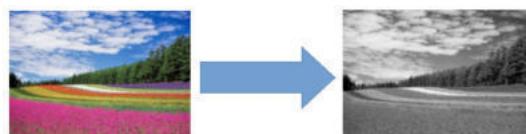
EVA-Prinzip



(Daten-)Verarbeitung, Definition

Datenverarbeitung hat die Ziele:

- Daten zu verändern

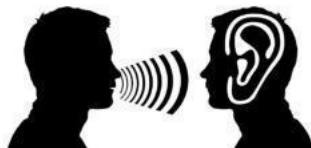


- Daten zu interpretieren



Information, Definition

Informationen sind Daten, die von einem Sender an einen Empfänger übermittelt werden.



Daten sind nur dann Informationen wenn der Empfänger diese vor dem Empfang nicht kannte.



Information, Definition

Welche Arten von Informationen oder Daten kennen Sie?

Beispiele:

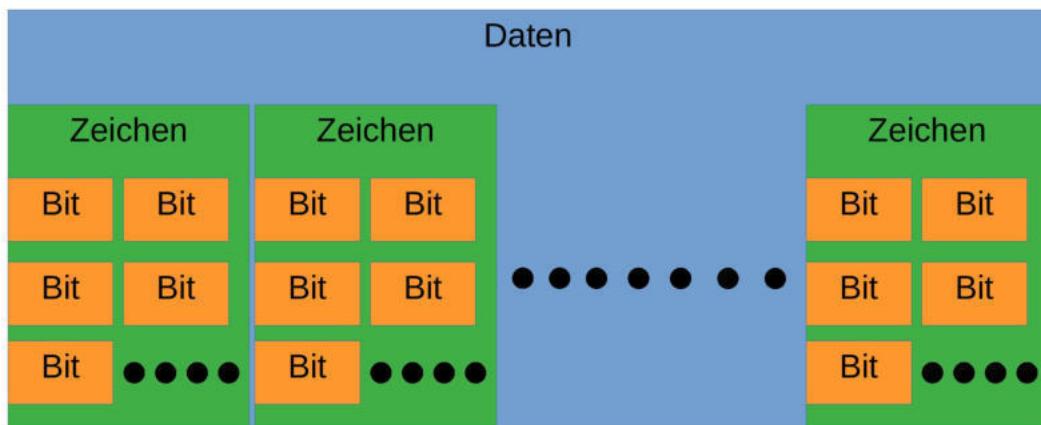
- Lottozahlen
-
-
-
-
-
-
-

Diese Aufzählung ist nicht abschließend!

Daten

Daten bestehen aus beliebig vielen Zeichen, mindestens aus einem Zeichen.

Zeichen bestehen aus beliebig vielen Bits., mindestens aus einem Bit.



Zeichen, Definition

In der Informatik besteht ein Zeichen aus einem oder mehreren Bits.

Ein Bit ist die kleinste Informationseinheit die es gibt. Ein Bit kann den Wert 1 oder 0 annehmen.

Das Zeichen A wird in der Informatik, genauer im ASCII Code, wie folgt dargestellt:

01000001_{bin}

Das Zeichen B:

01000010_{bin}

Daten, Definition

Definition aus der Wikipedia:

Für die **Datenverarbeitung** und (Wirtschafts-)Informatik werden Daten als **Zeichen** (oder Symbole) definiert, die **Informationen** darstellen und die dem Zweck der **Verarbeitung** dienen. (<https://de.wikipedia.org/wiki/Daten> abgerufen 13.09.2021)



Thema

