

PROGRAMMIEREN I

WS 2022

Prof. Dr. Kolja Eger
Hochschule für Angewandte Wissenschaften Hamburg

Check-In



Präsenzklausur im Labor (180min)

Dynamische Speicherallokation

Praktikum 7

Strukturen

Praktikum 6

Dateien

Praktikum 5

Zeiger

Praktikum 4

Vektoren (Arrays)

Praktikum 3

Funktionen

Praktikum 2

Kontrollstrukturen

Datentypen & Operatoren

Erste Programme

Praktikum 1



Unser Weg durch das Semester

Themen

Vom letzten Mal

- Erstes Programm
- Variablen
- Datentypen
 - Ganzzahlen
 - (Zeichen)
 - (Zeichenkette (Strings))

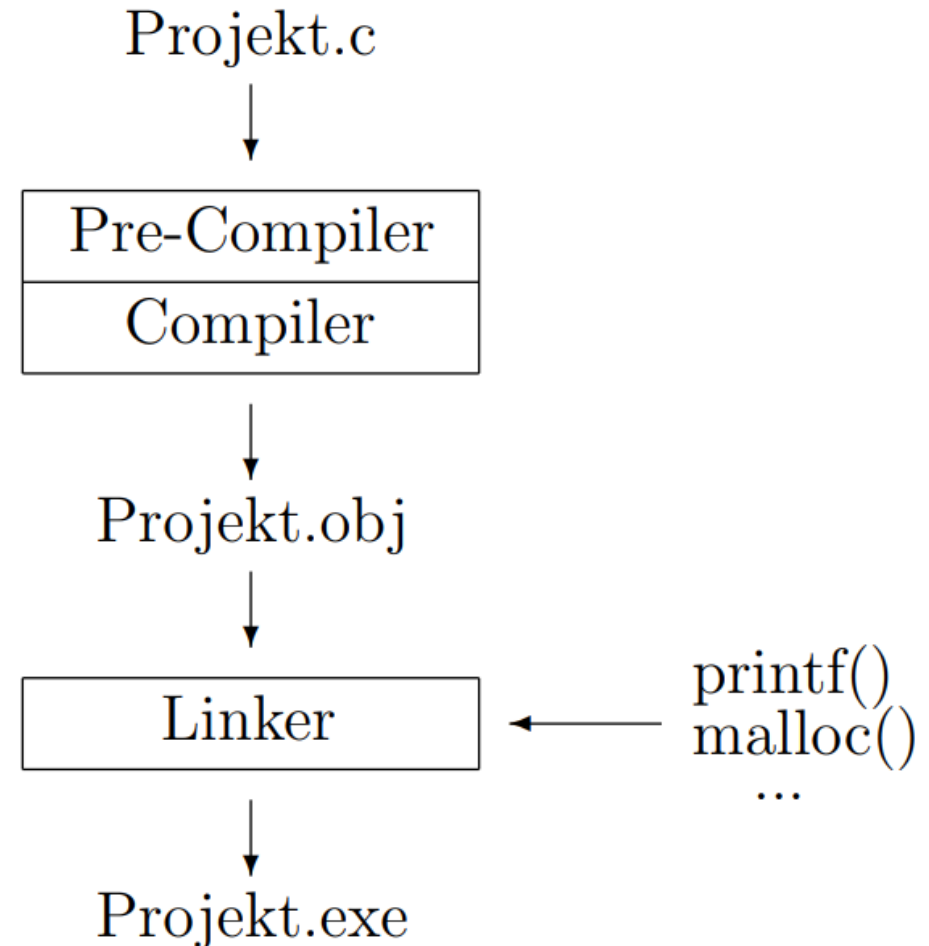
Von heute

- Compiler & CoLinker
- Ganzzahlen & Zeichen
- Operatoren
- Kontrollstrukturen
- Coding Style

COMPILER & CO

Wie wird aus Quellcode eine ausführbare Datei?

- Wenn Sie in Visual Studio ihren Code geschrieben haben und auf Starten drücken, passieren viele Schritte im Hintergrund
- eine vereinfachte Darstellung →
- Schritt 1: Aktionen vor dem Übersetzen durch den **Pre-Compiler**
- Schritt 2: Übersetzen des Quellcodes durch den **Compiler**
- Schritt 3: Zusammenfügen aller Funktionen durch den **Linker**



Wie wird aus Quellcode eine ausführbare Datei? (II)

- Schritt 1: Aktionen vor dem Übersetzen durch den **Pre-Compiler**
- Schritt 2: Übersetzen des Quellcodes durch den **Compiler**
- Schritt 3: Zusammenfügen aller Funktionen durch den **Linker**

Pre-Compiler (Preprozessor)

- Pre-Compiler (Vor-Übersetzer) erledigt einige Aufgaben bevor die Übersetzung beginnt
- Pre-Compiler-Befehle sind in C alle Befehle die mit einem Doppelkreuz **#** beginnen

Beispiel: `#include <include-Datei>` Pre-Compiler ersetzt alle include-Anweisungen durch den Inhalt der angegebenen Datei (vom Typ `.h`)

- Weitere Beispiele folgen im Semester (z.B. Makrodefinition, bedingte Übersetzung)

Compiler

- Compiler übersetzt den vom Pre-Compiler vorbereiteten Quelltext in eine für den Prozessor verständliche Sprache → Objekt-Datei mit Endung *.obj*
- In dieser Datei befindet sich nur die Übersetzung des Quellcodes
- Funktionen (z.B. *printf()*) aus Bibliotheken (include-Dateien) sind nicht Teil dieser Datei, sondern es ist nur der Aufruf eingefügt

Linker

- Linker geht Objekt-Datei durch und sucht alle verwendeten externen Funktionen
- Linker fügt den übersetzten Code für die benötigten Funktionen dem zu erstellenden Programm hinzu
- Der Linker erstellt eine ausführbare Datei, häufig mit Endung *.exe*

DATENTYPEN: GANZZAHLEN

Elementare Datentypen

| Typ | Beschreibung |
|--------|--|
| char | ganzzahliger Wert (ein Byte), bzw. ein Zeichen/Buchstabe (engl. <i>character</i>) |
| int | ganzzahliger Wert in der auf dem Rechner 'natürlichen' Größe |
| float | Gleitkommazahl mit einfacher Genauigkeit |
| double | Gleitkommazahl mit doppelter Genauigkeit |

Ganzzahlige Datentypen

- Elementare ganzzahlige Datentypen sind `int` und `char`
- `char` kann ein Zeichen/Buchstaben speichern bzw. stellvertretend eine kleine Zahl
- Neben `int` können auch kleinere/größere Ganzzahl definiert werden:
 - Kleinere (min. 2 Bytes): `short int` (oder in kurz) `short`
 - Größere (min. 4 Bytes): `long int` bzw. `long`
- Den Datentypen `char`, `short`, `int` und `long` kann das Wort `signed` oder `unsigned` vorgestellt werden
- Dies legt fest, ob sich der Wertebereich auf positive & negative oder nur auf positive Zahlen beschränkt
- `short`, `int` und `long` sind ohne Angabe vorzeichenbehaftet (`char` nicht)
- Ist nur `signed` oder `unsigned` angegeben, handelt es sich um den Datentyp `int`

Ganzzahlige Datentypen - Beispiele

```
3  int main() {  
4      short a;      // short integer (mit Vorzeichen)  
5      unsigned b;    // positiver int  
6      unsigned short c; // short integer (ohne Vorzeichen)  
7      signed long d;  // long integer (mit Vorzeichen)  
8      signed char e;  // char mit Vorzeichen (Wertebereich: -128 bis 127)  
9  }
```

Wie werden Ganzzahlen interpretiert?

- Standardmäßig werden alle Zahlen als `int` interpretiert
- Aber die Angabe in anderen Datentypen ist auch möglich, z.B.
 - Typ `long` wird durch „l“ oder „L“ erzeugt
 - Vorzeichenlose Zahl: „u“ oder „U“
 - Kombination möglich „ul“

Beispiele:

1234

1234L

1234u

1234ul

Datentyp `char`

- Steht sowohl für eine kleine Zahl als auch für ein Zeichen
- D.h. einer Variablen vom Typ `char` können Sie ein Zeichen zuweisen (einfache Anführungszeichen!)
- Der numerische Wert ist im ASCII-Code festgelegt
 - ASCII = American Standard Code for Information Interchange
 - Definition von 128 Zeichen (Code 0 bis 127) (siehe nä. Slide)
- C unterscheidet nicht zwischen Zeichen und Ganzzahlen, da Zeichen intern als Ganzzahlen dargestellt werden
- Sie können mit Zeichen rechnen!
- Sonderzeichen wird ein `\` vorangestellt, z.B. „`\n`“ für Zeilenvorschub

```
char a = 'B';
```

ASCII-Code von 'B' ist 66

```
char a = 'A' + 1;
```

ASCII-Tabelle

Sonderzeichen,
z.B. ESC = Escape,

| Dez/Hex/Okt | Zeichen | Dez/Hex/Okt | Zeichen | Dez/Hex/Okt | Zeichen | Dez/Hex/Okt | Zeichen |
|-------------|---------|-------------|---------|-------------|---------|-------------|---------|
| 0/00/000 | NUL | 32/20/040 | SP | 64/40/100 | @ | 96/60/140 | ` |
| 1/01/001 | SOH | 33/21/041 | ! | 65/41/101 | A | 97/61/141 | a |
| 2/02/002 | STX | 34/22/042 | " | 66/42/102 | B | 98/62/142 | b |
| 3/03/003 | ETX | 35/23/043 | # | 67/43/103 | C | 99/63/143 | c |
| 4/04/004 | EOT | 36/24/044 | \$ | 68/44/104 | D | 100/64/144 | d |
| 5/05/005 | ENQ | 37/25/045 | % | 69/45/105 | E | 101/65/145 | e |
| 6/06/006 | ACK | 38/26/046 | & | 70/46/106 | F | 102/66/146 | f |
| 7/07/007 | BEL | 39/27/047 | ' | 71/47/107 | G | 103/67/147 | g |
| 8/08/010 | BS | 40/28/050 | (| 72/48/110 | H | 104/68/150 | h |
| 9/09/011 | TAB | 41/29/051 |) | 73/49/111 | I | 105/69/151 | i |
| 10/0A/012 | LF | 42/2A/052 | * | 74/4A/112 | J | 106/6A/152 | j |
| 11/0B/013 | VT | 43/2B/053 | + | 75/4B/113 | K | 107/6B/153 | k |
| 12/0C/014 | FF | 44/2C/054 | , | 76/4C/114 | L | 108/6C/154 | l |
| 13/0D/015 | CR | 45/2D/055 | - | 77/4D/115 | M | 109/6D/155 | m |
| 14/0E/016 | SO | 46/2E/056 | . | 78/4E/116 | N | 110/6E/156 | n |
| 15/0F/017 | SI | 47/2F/057 | / | 79/4F/117 | O | 111/6F/157 | o |
| 16/10/020 | DLE | 48/30/060 | 0 | 80/50/120 | P | 112/70/160 | p |
| 17/11/021 | DC1 | 49/31/061 | 1 | 81/51/121 | Q | 113/71/161 | q |
| 18/12/022 | DC2 | 50/32/062 | 2 | 82/52/122 | R | 114/72/162 | r |
| 19/13/023 | DC3 | 51/33/063 | 3 | 83/53/123 | S | 115/73/163 | s |
| 20/14/024 | DC4 | 52/34/064 | 4 | 84/54/124 | T | 116/74/164 | t |
| 21/15/025 | NAK | 53/35/065 | 5 | 85/55/125 | U | 117/75/165 | u |
| 22/16/026 | SYN | 54/36/066 | 6 | 86/56/126 | V | 118/76/166 | v |
| 23/17/027 | ETB | 55/37/067 | 7 | 87/57/127 | W | 119/77/167 | w |
| 24/18/030 | CAN | 56/38/070 | 8 | 88/58/130 | X | 120/78/170 | x |
| 25/19/031 | EM | 57/39/071 | 9 | 89/59/131 | Y | 121/79/171 | y |
| 26/1A/032 | SUB | 58/3A/072 | : | 90/5A/132 | Z | 122/7A/172 | z |
| 27/1B/033 | ESC | 59/3B/073 | ; | 91/5B/133 | [| 123/7B/173 | { |
| 28/1C/034 | FS | 60/3C/074 | < | 92/5C/134 | \ | 124/7C/174 | |
| 29/1D/035 | GS | 61/3D/075 | = | 93/5D/135 |] | 125/7D/175 | } |
| 30/1E/036 | RS | 62/3E/076 | > | 94/5E/136 | ^ | 126/7E/176 | ~ |
| 31/1F/037 | US | 63/3F/077 | ? | 95/5F/137 | _ | 127/7F/177 | DEL |

Groß- &
Kleinbuchstaben
(keine Umlaute)

Sonderzeichen im Überblick

| | |
|--|--|
| <code>\n</code> Zeilenvorschub | <code>\r</code> Wagenrücklauf |
| <code>\t</code> Tabulator | <code>\v</code> Vertikaltabulator |
| <code>\b</code> backspace | <code>\f</code> Seitenvorschub |
| <code>\'</code> Einfachanführungszeichen | <code>\"</code> Doppelanführungszeichen |
| <code>\a</code> Klingelzeichen | <code>\?</code> Fragezeichen |
| <code>\ooo</code> ASCII-Code (oktal) | <code>\xhh</code> ASCII-Code (hexadezimal) |
| <code>\\</code> Gegenstrich | |

Zeichenkette (Strings)

- Eine Zeichenkette wird durch doppelte Anführungszeichen angedeutet und kann beliebige Anzahl von Zeichen enthalten
- Es gelten die gleichen Sonderzeichen
- Zeichenketten können auch leer sein
- Aufeinanderfolgende Zeichenketten werden als eine interpretiert und können so auf mehrere Zeilen aufgeteilt werden
- Eine Zeichenkette ist ein Vektor von Zeichen → Vektoren werden wir noch in einer anderen Vorlesung behandeln
- Intern wird eine Zeichenkette durch null (`\0`) begrenzt. Deshalb benötigt eine Zeichenkette im Speicher ein Byte mehr als die Anzahl der Zeichen
- Beachte den Unterschied:
 - `'x'` → ein Zeichen
 - `"x"` → eine Zeichenkette mit null-Terminierung

Zeichenkette

Sonderzeichen

```
printf("Hello world\n");
```

```
printf("");
```

```
printf("Hello "  
      "World" "\n");
```

→ Beispiel in Visual Studio

```
#include <stdio.h>
int main()
{
    char ch1 = 'x';
    char ch2 = 66;
    char ch3 = 'A' + 2;;

    // Ausgabe als Zeichen
    printf("%c %c %c\n", ch1, ch2, ch3);

    // Ausgabe als Zahlen
    printf("%d %d %d\n", ch1, ch2, ch3);

    // Sonderzeichen
    printf("\"\\t\\'\\t\\\\t\\?\\n");

    // Klingelzeichen
    printf("\\a");

    // Zeichenkette über mehrere Zeilen
    printf("Hello "
           "World" "\\n");

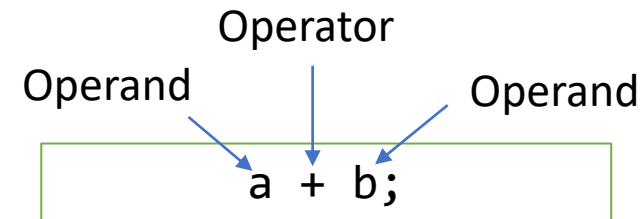
    return 0;
}
```

OPERATOREN

Operatoren

- Ein Operator verarbeitet Operanden
- *Unäre*, *binäre* und *ternäre* Operatoren

- Unäre erwarten ein Operand
- Binäre → zwei
- Ternäre → drei



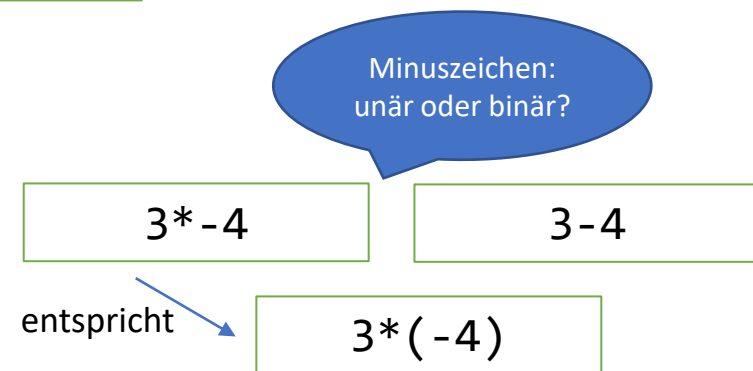
`-a;`

`a + b;`

`a < b ? a : b;`

Details in der
nä. Vorlesung

- Unterscheidung zwischen gleichnamigen Operanden ergibt sich aus dem Kontext



Arithmetische Operatoren

- 5 arithmetische Operatoren
 - Vier Grundrechenarten
 - Restdivision % (Beispiel: $17 \% 5 = 2$)
- Können auf Ganzzahl- und Gleitkommatypen angewandt werden
 - **Bereichsüberschreitungen** verhindern! (Verhalten systemabhängig)
 - **Division mit 0** verhindern! (führt zu einem Fehler)
- Restdivision nur auf Ganzzahlen (für negative Zahlen systemabhängig/undefiniert)
- Unäre Operatoren + und – (Vorzeichen) haben Vorrang ggü. Punktrechnung

$a + b;$

$a - b;$

$a * b;$

$a / b;$

$a \% b;$

Vergleichsoperatoren

- Das Ergebnis eines Vergleichsoperators ergibt *wahr* oder *nicht wahr*
 - Beispiel: `3==4` ergibt *nicht wahr* während `5<8` *wahr* ergibt
- Arithmetische Operatoren haben einen höheren Rang als Vergleichsoperatoren (d.h. sie werden vor dem Vergleich durchgeführt)
 - Beispiel: `counter < max-1` ist wie `counter < (max-1)`
- Die Operatoren `==` und `!=` haben einen geringeren Rang als die anderen Vergleichsoperatoren
 - Beispiel: `3<4==2>1` ist wie `(3<4)==(2>1)`
- Ergebnisse können auch als *eins* oder *null* interpretiert werden
 - Welchen Wert hat a? `a = (1 > 2 == 1);`

6 Vergleichsoperatoren in C

| | |
|--------------------|--------------------|
| <code>==</code> | gleich wie |
| <code>!=</code> | ungleich wie |
| <code><</code> | kleiner als |
| <code>></code> | größer als |
| <code><=</code> | kleiner gleich als |
| <code>>=</code> | größer gleich als |

Verknüpfungs-/Logikoperatoren

- Wie ermittle ich, dass zwei oder mehr Vergleiche gleichzeitig zutreffen?
 - Mit logischen Verknüpfungsoperatoren
 - `&&` ist eine UND-Verknüpfung
 - `||` ist eine ODER-Verknüpfung
 - Operatoren erwarten zwei logische Werte und geben einen logischen Wert zurück, also *wahr* oder *nicht wahr*
 - Bei UND müssen beide Werte wahr sein, damit das Ergebnis auch wahr ist; bei ODER nur einer der Werte (→ nä. Slide)
- Operatoren `&&` und `||` können auch auf Zahlen angewandt werden
 - 0 wird als nicht wahr interpretiert
 - Alle anderen Zahlen als wahr
- Rang der Verknüpfung geringer als Vergleichsoperatoren
- Rang von ODER ist geringer als von UND

Beispiel:

```
printf("%d\n", 0 && 0);  
printf("%d\n", 0 && 1);  
printf("%d\n", 1 && 0);  
printf("%d\n", 1 && 1);  
printf("\n");  
printf("%d\n", 0 || 0);  
printf("%d\n", 0 || 1);  
printf("%d\n", 1 || 0);  
printf("%d\n", 1 || 1);
```

Ausgabe: ?

Wahrheitstabelle der Verknüpfungsoperatoren

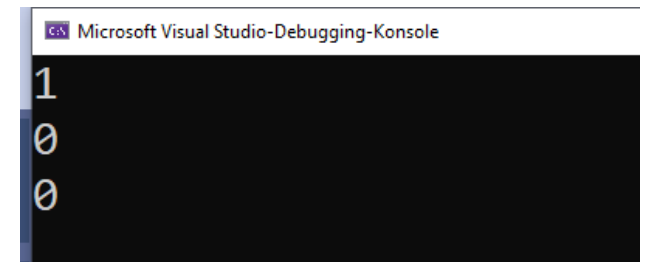
| linker Operand | rechter Operand | (oder) | && (und) |
|----------------|-----------------|------------|------------|
| nicht wahr | nicht wahr | nicht wahr | nicht wahr |
| nicht wahr | wahr | wahr | nicht wahr |
| wahr | nicht wahr | wahr | nicht wahr |
| wahr | wahr | wahr | wahr |

Negationsoperator

- *Wahr* wird zu *nicht wahr*
- Der unäre Negationsoperator wird durch das Ausrufezeichen *!* dargestellt
- Es negiert (d.h. kehrt um) den logischen Werte eines Ausdruck
- Auf Zahlen angewandt wird die Null (*nicht wahr*) zu einer 1 (*wahr*)
 - Alle anderen Zahlen (auch negative) werden zu einer 0 (*nicht wahr*)

```
printf("%d\n", !0);  
printf("%d\n", !1);  
printf("%d\n", !2);
```

Ausgabe:



```
Microsoft Visual Studio-Debugging-Konsole  
1  
0  
0
```

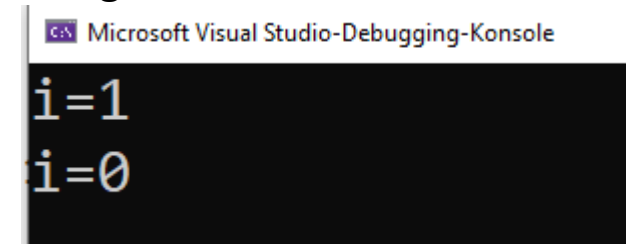
Inkrement und Dekrement

- Es gibt zwei unäre Operatoren zum Inkrementieren und Dekrementieren von ganzen Zahlen
 - `++` → Inkrementoperator erhöht eine ganze Zahl um eins
 - `--` → Dekrementoperator erniedrigt eine ganze Zahl um eins
- Operatoren können nur auf Variablen angewandt werden
 - Nicht erlaubt! `7++` oder `(a+b)++`
- Operator kann in Präfix- oder Postfix-Notation angewandt werden
 - Bei Präfix wird der Operator vor dem Operand geschrieben, z.B. `++i`
 - Dies bewirkt, dass erst der Operator angewandt und dann das Ergebnis zurückgegeben wird
 - Bei Postfix andersherum, z.B. `i++`

Beispiel:

```
int i = 0;
i++;
printf("i=%d\n", i);
i--;
printf("i=%d\n", i);
```

Ausgabe:



```
Microsoft Visual Studio-Debugging-Konsole
i=1
i=0
```

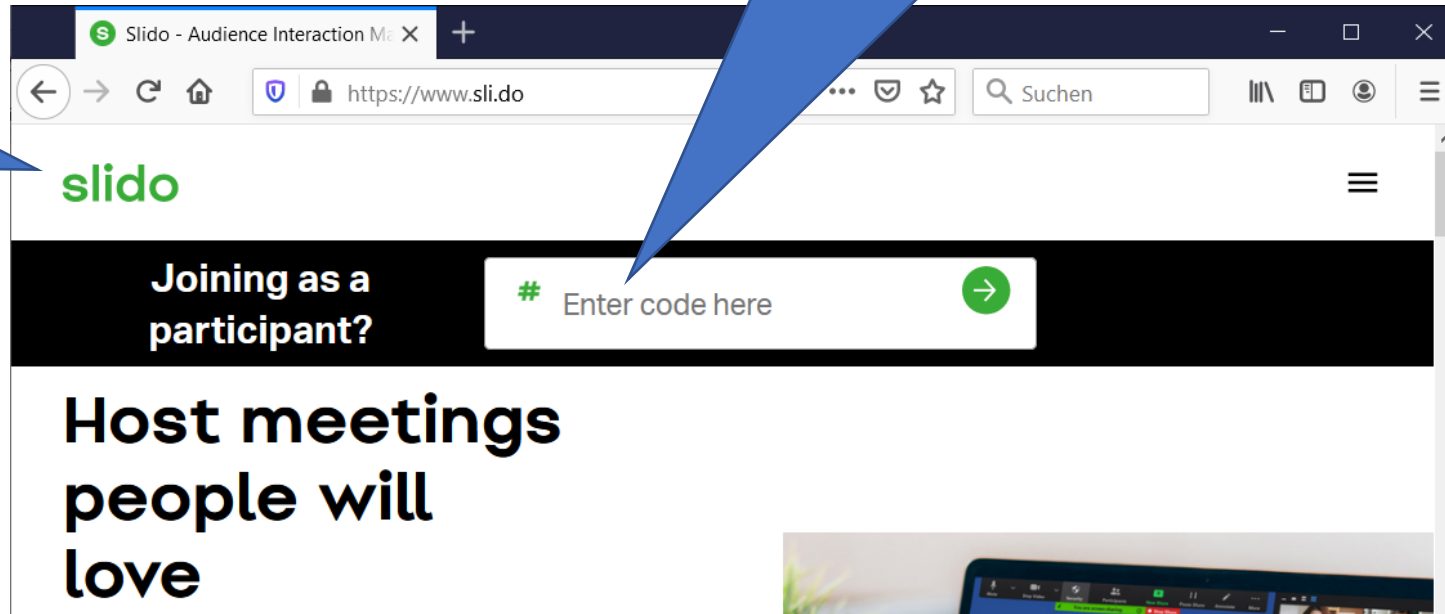
```
int i = 0;
printf("i=%d\n", i++);
printf("i=%d\n", --i);
```

Ausgabe: ?

Quiz

Slido.com

HAW-PR1



SCHNELLER ÜBERBLICK ZU KONTROLLSTRUKTUREN

In späteren Vorlesungen weitere Details..

If-else-Verzweigung

```
if (Bedingung) { Aktion }  
else { Aktion }
```

Beispiel:

```
#include <stdio.h>  
  
int main() {  
    int Kontostand = 345;  
    if (Kontostand >= 0) {  
        printf("Im gruenen Bereich\n");  
    }  
    else {  
        printf("Im Dispo\n");  
    }  
}
```

While-Schleife

while (Schleifenbedingung) { Aktion }

Beispiel:

```
#include <stdio.h>

int main() {
    double Kontostand = 345.23;
    while (Kontostand >= 50) {
        Kontostand = Kontostand - 50;
        printf("50 Euro ausgegeben. Neuer Kontostand= %.2f\n", Kontostand);
    }
}
```

For-Schleife

for (Initialisierung; Schleifenbedingung; Änderung) {Aktion}

Beispiel:

```
int i = 0;
Kontostand = 345.23;
for (i = 0; i < 5; i++) {
    Kontostand = Kontostand - 50;
    printf("50 Euro ausgeben. Neuer Kontostand= %.2f\n", Kontostand);
}
```

CODING STYLE

Coding Style

- Es gibt kein Programmierer & keine Programmiererin, der/die nicht auch Code von anderen benutzt
- Auch wenn man Code nach Monaten wieder liest, möchte man ihn schnell verstehen
- Coding Style (oder auch Programmierstil) beschreibt Regeln, wie Code gestaltet wird, z.B. Kommentare, aussagekräftige Bezeichner, Einrücken des Codes, ..
- Vorteile
 - Einheitlicher Programmierstil erleichtert die Einarbeitung
 - Programme sind einfacher zu lesen & zu verstehen
 - Programme lassen sich leichter in eine andere Umgebung portieren
 - Typische Fehler werden vermieden
 - Mehrere Programmierer können leichter an einem Programm arbeiten

Coding Style

- Coding Style ist nicht objektiv, sondern eine subjektive Entscheidung des Entwicklerteams
- Es können Best Practices verwendet werden oder Firmen definieren ihren eigenen Coding Style (z.B. [Linux Kernel Coding Style](#) oder [Google C++ Coding Style](#))
- In dieser Veranstaltung nutzen wir den [Coding Style von Prof. Heß](#)
- Für die erste Praktikumsaufgabe bitte beachten
 - Datei mit Kommentarkopf, welche Name, Beschreibung, Autor, Datum und Version angibt
 - Code sauber einrücken (mit TAB)
 - Alle angelegten Variablen in separaten Zeilen mit einem kurzen Kommentar rechts daneben
 - Den Rest des Quellcodes in Absätzen mit Kommentaren oberhalb gruppieren

Coding Style - Beispiel

```
// Kontostand prüfen
// Beispielprogramm für if-Anweisung
// Autor: Kolja Eger
// Datum: 8.3.2022
// Version: 1.0

#include <stdio.h>
int main() {
    double Kontostand = 345.23;    // Kontostand in Euro

    // Info an Benutzer
    printf("Beispiel if-else-Anweisung-Schleife -> Kontostand pruefen\n");

    // Prüfe, ob Kontostand positiv oder negativ
    if (Kontostand >= 0)
        printf("Im gruenen Bereich\n"); // Ausgabe bei positiven Kontostand
    else
        printf("Im Dispo\n");    // Ausgabe bei negativen Kontostand

    return 0;
}
```

Aufgabe/Demo Praktikum 1

BEISPIELPROGRAMM: UMRECHNUNG FAHRENHEIT → CELSIUS

→ Beispiel in Visual Studio

```
// Umwandlung von Fahrenheit in Celsius
```

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int fahr, celsius;
```

```
    int lower, upper, step;
```

```
    lower = 0;
```

```
    /* untere Grenze der Temperatur */
```

```
    upper = 300;
```

```
    /* obere Grenze */
```

```
    step = 20;
```

```
    /* Schrittweite */
```

```
    /* Ausgabe einer Überschrift */
```

```
    printf("Fahr.\tCelsius\n");
```

```
    /* Erstellung der Tabelle */
```

```
    fahr = lower;
```

```
    while (fahr <= upper) {
```

```
        celsius = 5 * (fahr - 32) / 9;
```

```
        printf("%d\t%d\n", fahr, celsius);
```

```
        fahr = fahr + step;
```

```
    }
```

```
    return 0;
```

```
}
```

Mehrere Variablen vom gleichen Typ können auch in einer Zeile definiert werden

Sonderzeichen für TAB

Schleifen können ähnlichen Code wiederholen, wo sich z.B. nur Variablen ändern

Das Innere der Schleife wird solange durchgeführt bis die Schleifenbedingung nicht mehr erfüllt ist

Achtung, typischer Fehler: Endlos-Schleife – Schleifenbedingung ist immer „wahr“

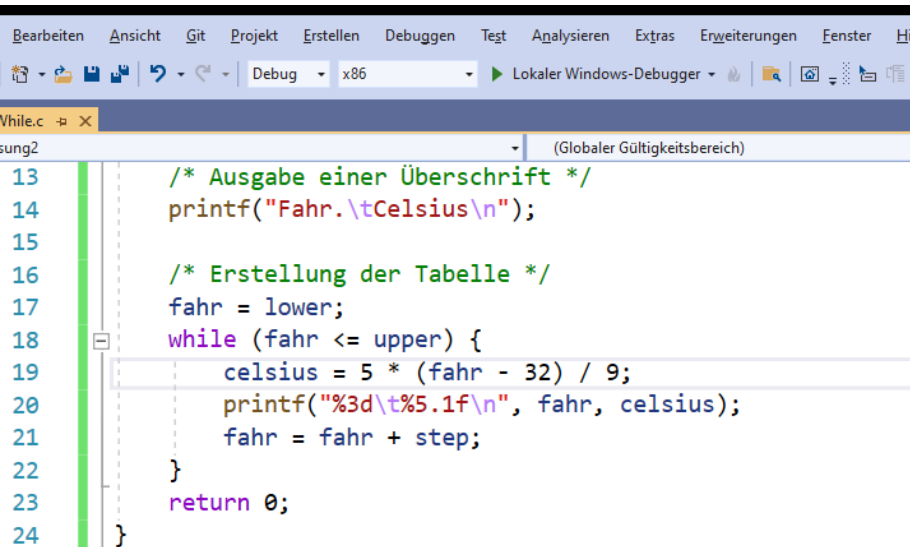
Auch hier:
Punkt vor Strichrechnung!

Mehrere Platzhalter (%d) möglich!

Debugger in Visual Studio

Beim Ausführen im Debug-Modus wird an Haltepunkten gestoppt (→ gelber Pfeil)

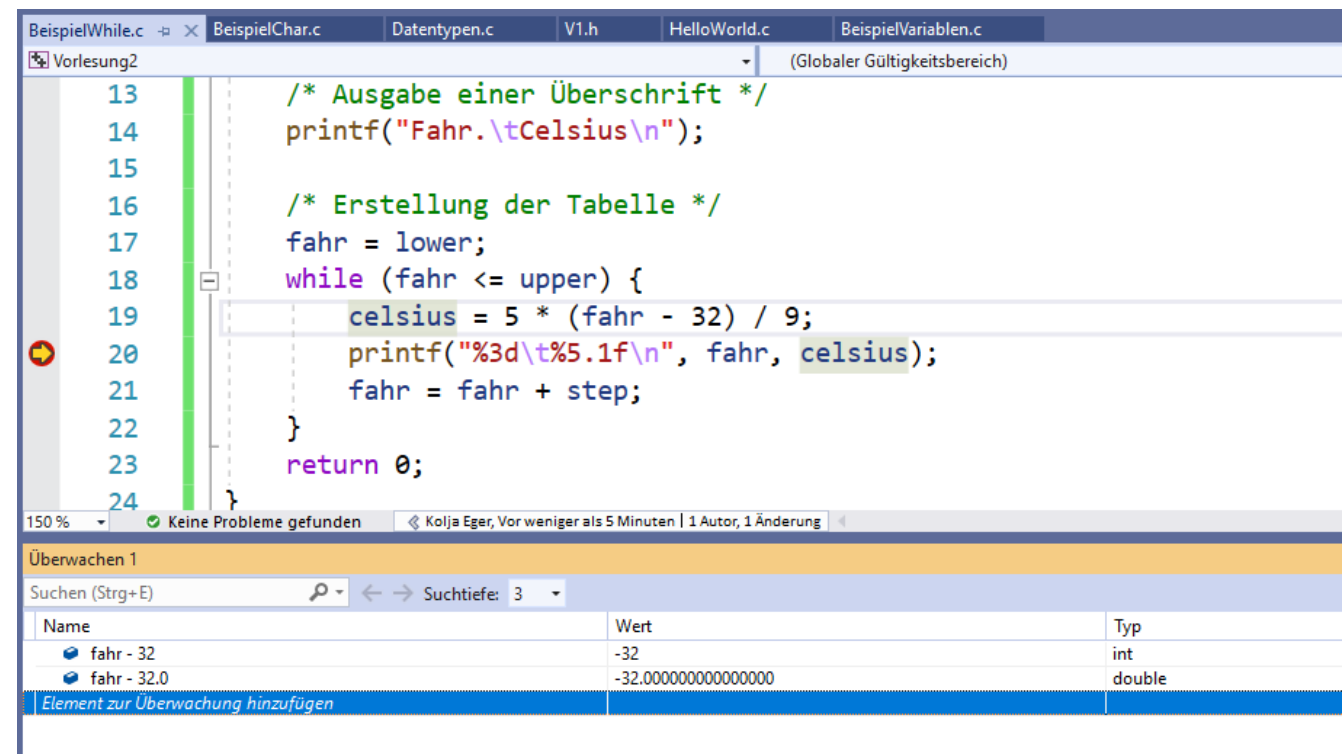
Und Sie können in *Auto*, *Lokal* und *Überwachen* Werte von Variablen einsehen und Ausdrücke berechnen



```
13  /* Ausgabe einer Überschrift */
14  printf("Fahr.\tCelsius\n");
15
16  /* Erstellung der Tabelle */
17  fahr = lower;
18  while (fahr <= upper) {
19      celsius = 5 * (fahr - 32) / 9;
20      printf("%3d\t%5.1f\n", fahr, celsius);
21      fahr = fahr + step;
22  }
23  return 0;
24  }
```

A red dot indicates a breakpoint set on line 18.

Haltepunkte im Code setzen:
Klicken Sie mit der Maus in den grauen Bereich in einer Zeile →
Ein roter Punkt erscheint



```
13  /* Ausgabe einer Überschrift */
14  printf("Fahr.\tCelsius\n");
15
16  /* Erstellung der Tabelle */
17  fahr = lower;
18  while (fahr <= upper) {
19      celsius = 5 * (fahr - 32) / 9;
20      printf("%3d\t%5.1f\n", fahr, celsius);
21      fahr = fahr + step;
22  }
23  return 0;
24  }
```

A yellow arrow points to line 20, indicating the current execution point. Below the code editor, the 'Überwachen' (Watch) window is open, showing the following table:

| Name | Wert | Typ |
|------------------------------------|----------------------|--------|
| fahr - 32 | -32 | int |
| fahr - 32.0 | -32.0000000000000000 | double |
| Element zur Überwachung hinzufügen | | |

→ Beispiel jetzt mit for-Schleife

```
// Umwandlung von Fahrenheit in Celsius
#include <stdio.h>
int main()
{
    int fahr;
    float celsius;
    int lower, upper, step;

    lower = 0;           /* untere Grenze der Temperatur */
    upper = 300;         /* obere Grenze */
    step = 20;           /* Schrittweite */

    /* Ausgabe einer Überschrift */
    printf("Fahr.\tCelsius\n");

    /* Erstellung der Tabelle */
    for (fahr = lower; fahr <= upper; fahr = fahr + step) {
        celsius = 5.0 / 9.0 * (fahr - 32.0);
        printf("%6d\t%6.1f\n", fahr, celsius);
    }
    return 0;
}
```

Umrechnung der verschiedenen Werte kann auch mit for-Schleife umgesetzt werden

Initialisierung, Schleifenbedingung und Änderung stehen im Kopf der for-Schleife



GLEITKOMMAZAHLEN

Elementare Datentypen

| Typ | Beschreibung |
|--------|--|
| char | ganzzahliger Wert (ein Byte), bzw. ein Zeichen/Buchstabe (engl. <i>character</i>) |
| int | ganzzahliger Wert in der auf dem Rechner 'natürlichen' Größe |
| float | Gleitkommazahl mit einfacher Genauigkeit |
| double | Gleitkommazahl mit doppelter Genauigkeit |

Gleitkommazahlen

- Drei Typen (im Allgemeinen)
 - Float, double und long double
 - Unterscheiden sich in Größe und Genauigkeit

`sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)`

- Genaue Umsetzung ist systemabhängig
- In Visual Studio:

| | Bytes | Wertebereich | |
|---------------------|-------|---------------------------|------------------------|
| <code>float</code> | 4 | 3.4E +/- 38 (7 Stellen) | → Einfache Genauigkeit |
| <code>double</code> | 8 | 1.7E +/- 308 (15 Stellen) | → Doppelte Genauigkeit |

<https://docs.microsoft.com/de-de/cpp/cpp/data-type-ranges?view=msvc-170>

→ Beispiel in Visual Studio. Aus int wird float

```
1 // Umwandlung von Fahrenheit in Celsius
2 #include <stdio.h>
3 int main()
4 {
5     int fahr, celsius;
6     //
7     int lower, upper, step;
8
9     lower = 0;        /* untere Grenze der Temperatur */
10    upper = 300;       /* obere Grenze */
11    step = 20;         /* Schrittweite */
12
13    /* Ausgabe einer Überschrift */
14    printf("Fahr.\tCelsius\n");
15
16    /* Erstellung der Tabelle */
17    fahr = lower;
18    while (fahr <= upper) {
19        celsius = 5 * (fahr - 32) / 9;
20        printf("%d\t%d\n", fahr, celsius);
21        fahr = fahr + step;
22    }
23    return 0;
24 }
```

Version 1 mit int

```
1 // Umwandlung von Fahrenheit in Celsius
2 #include <stdio.h>
3 int main()
4 {
5     int fahr;
6     float celsius;
7     int lower, upper, step;
8
9     lower = 0;        /* untere Grenze der Temperatur */
10    upper = 300;       /* obere Grenze */
11    step = 20;         /* Schrittweite */
12
13    /* Ausgabe einer Überschrift */
14    printf("Fahr.\tCelsius\n");
15
16    /* Erstellung der Tabelle */
17    fahr = lower;
18    while (fahr <= upper) {
19        celsius = 5 * (fahr - 32.0) / 9;
20        printf("%3d\t%5.1f\n", fahr, celsius);
21        fahr = fahr + step;
22    }
23    return 0;
24 }
```

Version 2 mit float

→ Beispiel in Visual Studio: Aus int wird float

```
// Umwandlung von Fahrenheit in Celsius
```

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int fahr;
```

```
    float celsius;
```

```
    int lower, upper, step;
```

Variable `celsius` ist jetzt eine Gleitkommazahl

```
    lower = 0;           /* untere Grenze der Temperatur */
```

```
    upper = 300;         /* obere Grenze */
```

```
    step = 20;           /* Schrittweite */
```

```
    /* Ausgabe einer Überschrift */
```

```
    printf("Fahr.\tCelsius\n");
```

```
    /* Erstellung der Tabelle */
```

```
    fahr = lower;
```

```
    while (fahr <= upper) {
```

```
        celsius = 5 * (fahr - 32.0) / 9;
```

```
        printf("%3d\t%5.1f\n", fahr, celsius);
```

```
        fahr = fahr + step;
```

```
    }
```

```
    return 0;
```

```
}
```

In C müssen sie bei gemischten Ausdrücken (float & int) aufpassen als was für Datentypen die Variablen bzw. Konstanten interpretiert werden im Vergleich zu dem Ergebnis was sie herausbekommen wollen!

Bei Ganzzahlen wird das Ergebnis einer Division abgeschnitten!

Mit 32.0 anstatt 32 sind Zwischenergebnisse jetzt vom Typ `float`

Platzhalter für Gleitkommazahlen ist `%f`
Mit `.1` geben wir die Nachkommastellen an
Mit `5` die Breite (=Anzahl der Zeichen) insgesamt

Und was
mach ich
bis zum
nächsten
Mal?



Beispiele aus Vorlesung
ausprobieren



Praktikum vorbereiten

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!