

## 2. Übung

- 1 Wiederholung
- 2 Arbeit mit SVN
- 3 Variablen
- 4 Aufgaben
- 5 Nutzung Debugger (Siehe Foliensatz 1)

# 1. Wiederholung

Verzeichnis (Imp2019) unter dem Laufwerksbuchstaben R: nutzen.

Wir haben kleine C- Programme geschrieben, die Lückentexte enthielten:

```
#include <stdio.h>
main(){

    printf(" Die Zahlen %d und %16.3f\n",9, 16.25);

}
```

Übersetzen und testen von Programmen

```
cd Imp2019
gcc -g anfang.c -o anfang.exe
anfang.exe
```

```
< --- wechsele ins Verzeichnis Imp2019
< --- führe dort C-Compiler aus
< --- starte das Programm
```

# 1. Wiederholung printf

## Formatierungen für printf und (scanf)

%d oder %i	Für Integerwerte
%X %x	Für Ausgabe in Hexadezimalsystem
%u	Für Integerwerte ohne Vorzeichen
%f	Floatwerte
%e	Floatwerte in Exponentialdarstellung
%c	Ausgabe von Buchstaben
%s	Ausgabe von Zeichenketten

# 1. Wiederholung printf

## Formatierungen von Ganzzahlen – Optionen

Für d und i

Beispiel %6d es werden immer 6 Stellen ausgegeben

Option	Beschreibung	Beispiele
-	Linksausrichtung der Zahl	%-6d
+	Ausgabe des Vorzeichen auch bei positiver Zahl	%+d %+8d
Width (Angabe einer Zahl)	Anzahl der Stellen, die ausgegeben werden. Wenn die Zahl weniger Stellen hat, dann Auffüllen mit Leerzeichen	%6d %8d
.precision (Angabe einer Zahl)	Minimale Anzahl ausgegebener Stellen Wenn die Zahl weniger Stellen hat, dann Auffüllen mit führenden Nullen	%.6d %.4d

# 1. Wiederholung printf

Formatierungen von Gleitkommazahlen – Optionen  
Für die e, E, f, F, g, G

Option	Beschreibung	Beispiele
-	Linksausrichtung der Zahl	%-10f
+	Ausgabe des Vorzeichen auch bei positiver Zahl	%+f %+12f
Width (Angabe einer Zahl)	Anzahl der Stellen, die ausgegeben werden. Wenn die Zahl weniger Stellen hat, dann Auffüllen mit Leerzeichen	%6f %8e
.precision (Angabe einer Zahl)	Für e, E, f: die Anzahl der Nachkommastellen Für g, G: die max ausgegebener Stellen oder Exponentialdarstellung	%10.2f %10.4g %10.4g

# 3. Variable

Bisher nur Ganzzahlen und Gleitpunktzahlen verwendet!

Welche weiteren Datentypen in Programmiersprache C?  
Wie kann man diese Zahlwerte speichern und wiederverwenden?

Programmiersprache C hat verschiedene Standards  
(K&R, ANSI-C, C99 )

Prozessoren sind heute leistungsfähiger, nicht 8-Bit- sondern schon bis zu 64-Bit Verarbeitungsbreite

Deshalb in Sprache nur festgelegt für die **Ganzzahldatentypen**  
die Relation **short <= int <= long** und  
für **Gleitpunktzahldatentypen float <= double**

# 3. Elementare Datentypen

Übliche Genauigkeiten in C Compilern (PC-Pool)

Ganze Zahlen:

- **char** (1 Byte-8-Bit) von -128 ... +127
- **short** (2 Byte-16-Bit) von -32768 ... +32767
- **int** (4 Byte-32-Bit) von -2147483648 ... +2<sup>31</sup> - 1
- **long** (4 Byte-32-Bit) von -2147483648 ... +2<sup>31</sup> - 1

Gleitkommazahlen:

- **float** (4 Byte-32-Bit) 1.17E-38 .. 3.4E38 (7-8 Stellen)
- **double** (8 Byte-64-Bit) 2.2E-308 .. 1.7E308 (15-16 Stellen)

*Beispiele zur Verwendung:*

```
int i;  
int j = 7;  
long zaehler = 1341;  
long int _zaehler = 1341367; // short int oder long int  
double laenge = 42.195;  
float lang = 42.195f;
```

# 3. Elementare Datentypen \*- Zusatz

## Genauigkeiten in 64-Bitarchitekturen für C Compiler

Compiler-Hersteller sind vorsichtig bei Wechsel der Standarddatentypen.

Da in Standardisierung auch die Bezeichnungen angedacht waren:

short int – int – long int

und damit Kombinationen von short und long möglich, wurden zusätzlich eingeführt:

64 – Bit Ganzzahl **long long** und

64 – Bit Gleitpunktzahl **long double**

Ganze Zahlen:

- ...
- **long** (4 Byte-32-Bit) von  $-2147483648 \dots +2^{31} - 1$
- **long long** (8 Byte-64-Bit) von  $-9223372036854775808 \dots +2^{63} - 1$   
(92233720368547758087)  
(Hinweis: in Vorlesungsfolie schon long mit 8 byte, da auf Mac und Linux)

Gleitkommazahlen:

- **float** (4 Byte-32-Bit)  $1.17\text{E}-38 \dots 3.4\text{E}38$  (7-8 Stellen)
- **double** (8 Byte-64-Bit)  $2.2\text{E}-308 \dots 1.7\text{E}308$  (15-16 Stellen)
- **long double** (12 Byte-64-Bit Mantisse)  $3.362103\text{E}-4932 \dots 1.89731\text{E}4932$

**Achtung im PC-Pool gibt es Probleme bei der Ausgabe von Variablen der 64-Bit Datentypen, da die minGW Implementation diese so nicht standardmäßig unterstützt.**



# 3. Elementare Datentypen

Interne Darstellung von char eines einzelnen Zeichens.

```
char c = 'K';
```

In 8-Bit können als maximaler Code die Potenzen von 2 von 0 bis 7 kombiniert werden

$2^0+2^1+2^2+2^3+2^4+2^5+2^6$  und statt der Position  $2^7$  das Vorzeichen gespeichert werden.

Maximaler Wert :

$$2^0+2^1+2^2+2^3+2^4+2^5+2^6+2^7=1+2+4+8+16+32+64+128=255$$

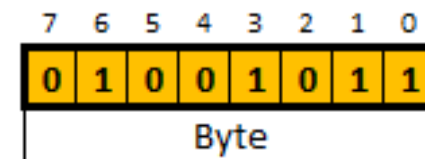
Oder  $+ 2^0+2^1+2^2+2^3+2^4+2^5+2^6=1+2+4+8+16+32+64=+127$

$- 2^0+2^1+2^2+2^3+2^4+2^5+2^6=1+2+4+8+16+32+64=(128 \text{ Werte } -0+127)$

In der C üblichen Kodierungstabelle ASCII hat der Große Buchstabe K den dezimalen Wert 75.

$$75 = 64+8+2+1 = 2^6+2^3+2^1+2^0$$

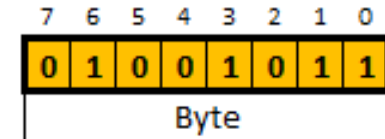
$$75 = 1*2^6+0*2^5+0*2^4+1*2^3+0*2^2+1*2^1+1*2^0$$



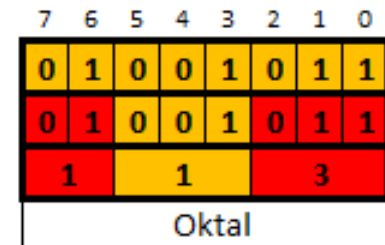
# 3. Elementare Datentypen

Interne Darstellung von char

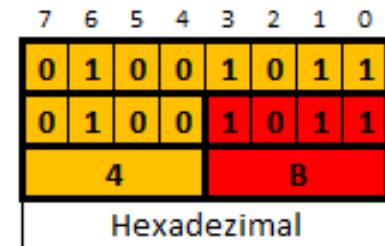
```
char c = 'K';
```



Da die Bit-Darstellungen recht lang werden, gibt es Zahlssysteme, die 3-Bit zusammenfassen Oktalzahlen  
( Ziffern 0, 1, 2, 3, 4, 5, 6, 7)



oder 4 Bit – Hexadezimalzahlzahlen  
(Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)



Im Beispiel der Buchstabe 'K', kann auch  
als Binärzahl 01001011,

als Dezimalzahl 75, // c = 75;

als Oktalzahl 113, // c = 0113; Zahlen mit Anfangsziffer 0!

als Hexadezimalzahl 4B // c = 0x4B;

beschrieben werden und hat damit immer den gleichen internen Wert.

# 3. Variable

Um mit Variablen in der Programmiersprache C arbeiten zu können, müssen eine Variable deklariert, definiert und initialisiert werden.

- **deklariert** – Name und Datentyp festlegen
- **definiert** – Speicherplatz reservieren
- **initialisiert** – Anfangswert zu weisen

```
int a; // Variable a deklariert, definiert und nicht initialisiert  
a = 3; // bestehende Variable a bekommt Wert 3 ( hier Initialisierung)
```

```
int b = 9; // Variable b deklariert, definiert und initialisiert mit 9
```

```
extern int c; // Variable c nur deklariert, hier keine Speicherung
```

# 3. Variable \*

Eine Variable hat immer einen Datentyp und einen Namen!

Ein Variable kann folgende Teile enthalten:

- eine Speicherklasse (**auto, register, static, extern**),
- einen Datentyp( **byte, short, int, long, long long, float, double, long double**),
- einen Zusatz, ob Datentyp mit oder ohne Vorzeichen (**unsigned, signed**)
- einen Namen (Frei wählbar (bis auf Ausnahmen)),
- einen Speicherplatz ( implizit durch Datentyp oder später wählbar bei Feldern),
- **einen Wert, der gespeichert wird.**

Variablen haben eine Gültigkeitsbereich ( meist Gültigkeitsbereich „Datei“ oder „Block“; Block - durch die Klammern { und } begrenzt).

Namen von Variablen – Regeln:

- Erstes Zeichen Buchstabe (englisches Alphabet) oder Zeichen \_
- Unterschied Groß- und Kleinschreibung
- Weitere Zeichen Buchstaben, Zahlen oder \_
- Keine Schlüsselworte der Programmiersprache zulässig
- Länge des Namens abhängig vom Sprachstandard

# 3. Beispiel

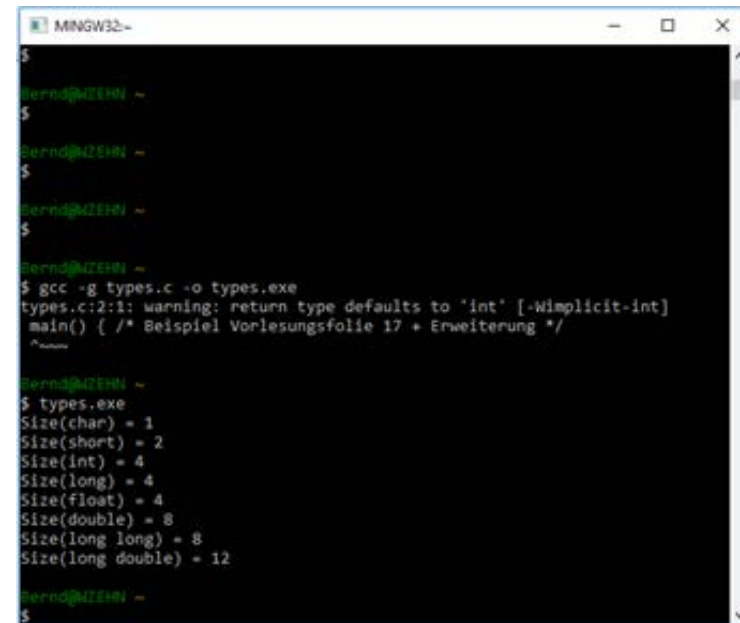
```
#include <stdio.h>
main() { /* Beispiel Vorlesungsfolie + Erweiterung */
```

```
    char a; short b; int c;
    long d; float e; double f;
    long long g; long double h;
    printf("Size(char) = %d\n", sizeof a);
    printf("Size(short) = %d\n", sizeof b);
    printf("Size(int) = %d\n", sizeof c);
    printf("Size(long) = %d\n", sizeof d);
    printf("Size(float) = %d\n", sizeof e);
    printf("Size(double) = %d\n", sizeof f);
```

```
    printf("Size(long long) = %d\n", sizeof g);
    printf("Size(long double) = %d\n", sizeof h);
```

```
}
```

Auf SVN-Server unter /public/src/win/types.c



```
MINGW32~
$
Bern@MTEHN ~
$
Bern@MTEHN ~
$
Bern@MTEHN ~
$
Bern@MTEHN ~
$ gcc -g types.c -o types.exe
types.c:2:1: warning: return type defaults to 'int' [-Wimplicit-int]
main() { /* Beispiel Vorlesungsfolie 17 + Erweiterung */
^~~~~~
Bern@MTEHN ~
$ types.exe
Size(char) = 1
Size(short) = 2
Size(int) = 4
Size(long) = 4
Size(float) = 4
Size(double) = 8
Size(long long) = 8
Size(long double) = 12
Bern@MTEHN ~
$
```

# 3. Variablen und deren Gültigkeit

```
#include <stdio.h>

int a;
int ggg= 40;
double d;
int _Dies_ist_EIN_langer_Variablenname = 2;
int _4Dies_3ist_3EIN_6langer_13Variablenname = 4;

main()
{
    int Ggg = 41;
    int GGg = 42;
    float f;
    double d;
    ggg = 40;
    {
        int a = 4;
        double d1 = 12.4;
        f = 12.4f;
        d = d1;
        printf("a=%d, ggg=%d, Ggg=%d, GGg=%d\n",a, ggg, Ggg, GGg);
    }
    printf(" Lange Variable =%d\n",_Dies_ist_EIN_langer_Variablenname);
    printf(" Lange Variable =%d\n",_4Dies_3ist_3EIN_6langer_13Variablenname);
    printf(" f=%f, und d=%f\n",f,d);
}
```

# 3. Variablen und deren Gültigkeit

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

```
int a;  
int ggg= 40;  
double d;  
int _Dies_ist_EIN_langer_Variablenname = 2;  
int _4Dies_3ist_3EIN_6langer_13Variablenname = 4;
```

Variablen in ganzer  
Quelltextdatei gültig

```
main()  
{
```

```
    int Ggg = 41;  
    int GGg = 42;  
    float f;  
    double d;  
    ggg = 40; a = 27;  
    {
```

```
        int a = 4;  
        double d1 = 12.4;  
        f = 12.4f;  
        d = d1;  
        printf("a=%d, ggg=%d, Ggg=%d, GGg=%d\n", a, ggg, Ggg, GGg);
```

Block

Block  
main

```
    }  
    printf(" Lange Variable =%d\n",_Dies_ist_EIN_langer_Variablenname);  
    printf(" Lange Variable =%d\n",_4Dies_3ist_3EIN_6langer_13Variablenname);  
    printf(" f=%f, und d=%f\n",f,d);
```

```
}
```

# 3. Variablen Rechenoperationen

Variablen können Werte ihres Datentyps speichern.

Die Werte können aus Berechnungen entstehen.

Rechenoperationen in C:

- Addition – Operationssymbol +
- Subtraktion – Operationssymbol -
- Multiplikation – Operationssymbol \*
- Division – Operationssymbol /
- Modulo – Operationssymbol %
- Potenz - Kein Operationssymbol!

```
int a = 2 + 4 - 3; // a speichert 3
int b = 4 * 5 ; // b speichert 20
int c = 19/5; // c speichert 3 (Ganzzahligen Division)
int d = 19%5; // d speichert 4 ( Rest bei Division)
```

```
double e = 19.0/5; // e speichert 3.8
double e = 19.0%5; // Compiler-Fehler invalid operand
```



# 3. Umwandlungen

```
#include <stdio.h>
/* Zuweisungen von möglicherweise falschen Zahlenbereichen zu Variablen */
main(){
    int a = 2;
    int b = 3.1; // Abschneiden auf 3
    int c = 12;
    int c1 = 012; // Oktalzahl 12 - Dezimal 10
    int c2 = 0xA; // Hexadezimalzahl A - Dezimal 10

    float f = 12.67f; // 12.67f in Float - Genauigkeit
    double d = 14.4; // Double-Genauigkeit
    double e = 16;
    printf(" a= %d, b=%d, c=%d, c1=%d, c2=%d, ", a, b, c, c1, c2);
    printf("f=%f, d=%f; e=%f\n", f, d, e);
    e = f;
    f = d;
    f = a;
    b = e;
    printf(" a= %d, b=%d, c=%d, c1=%d, c2=%d, ", a, b, c, c1, c2);
    printf("f=%f, d=%f; e=%f\n", f, d, e);
}
```

# 3. Umwandlungen

```
#include <stdio.h>
main(){
    int a = 2;
    int b; // Abschneiden auf 3
    double e;

    // Runden
    b = 3.1 + 0.5;
    printf(" Runden von 3.1= %d,",b);
    b = 3.6 + 0.5;
    printf(" Runden von 3.6= %d\n",b);

    // Vorsicht bei Berechnungen mit Ganzzahlen
    e = 1.0/3;
    printf(" Berechnung 1/3= %f,",e);
    e = 1/3;
    /* Die Berechnung erfolgt als Ganzzahl und dann wird das Ergebnis als
    Gleitpunktwert gespeichert. */
    printf(" Berechnung 1/3= %f\n",e);

}
```

# 3. Umwandlungen

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

MINGW32:~/Ipr2017

```
a= 2, b=3, c=12, c1=10, c2=10, f=12.670000, d=14.400000; e=16.000000  
a= 2, b=12, c=12, c1=10, c2=10, f=2.000000, d=14.400000; e=12.670000
```

Admin@gurke ~/Ipr2017

```
$ gcc -g ueb02_zuweis.c -o ueb02_zuweis.exe
```

Admin@gurke ~/Ipr2017

```
$ ueb02_zuweis.exe
```

```
a= 2, b=3, c=12, c1=10, c2=10, f=12.670000, d=14.400000; e=16.000000  
a= 2, b=12, c=12, c1=10, c2=10, f=2.000000, d=14.400000; e=12.670000  
Runden von 3.1= 3, Runden von 3.6= 4  
Berechnung 1/3= 0.333333, Runden von 3.6= 0.333333
```

Admin@gurke ~/Ipr2017

```
$ gcc -g ueb02_zuweis.c -o ueb02_zuweis.exe
```

Admin@gurke ~/Ipr2017

```
$ ueb02_zuweis.exe
```

```
a= 2, b=3, c=12, c1=10, c2=10, f=12.670000, d=14.400000; e=16.000000  
a= 2, b=12, c=12, c1=10, c2=10, f=2.000000, d=14.400000; e=12.670000  
Runden von 3.1= 3, Runden von 3.6= 4  
Berechnung 1/3= 0.333333, Berechnung 1/3= 0.000000
```

Admin@gurke ~/Ipr2017

```
$
```

# 4. Aufgaben

Schreiben Sie für mindestens einer der beiden folgenden Aufgaben ein Programm

1. Schreiben Sie ein Programm zur Berechnung des Body-Mass-Index  
(Formel:  $BMI = m / l^2$ , wobei  $m$  die Körpermasse (in Kilogramm) und  $l$  die Körpergröße (in m) angibt. Ausgangspunkt soll die Eingabe der Werte als Ganzzahlen sein, Die Masse wird in kg vorgegeben, die Körpergröße in cm) Bevor sie programmieren, fertigen sie ein Struktogramm an.)
2. Schreiben Sie ein Programm zur Berechnung der Schuhgröße  
(Unterschiedliche Systeme, teilweise historisch gewachsen, üblich z.B. Formeln  
Schuhgröße (EU) = (Fußlänge in cm + 1,5) × 1,5  
Brannock-System:  
Herrengröße (US) = Fußlänge in cm ÷ 2,54 × 3 – 22  
Damengröße (US) = Fußlänge in cm ÷ 2,54 × 3 – 21.  
Achtung die Schuhgröße in Europa ist eine Ganzzahl, im Brannocksystem sind Abstufungen von 0.5 möglich)

```
#include <stdio.h>
int main() { /* Eingaben und Ausgaben */
    int masse, laenge;
    printf("Eingabe der Koerpermasse in kg\n");
    scanf("%d",&masse);
    printf("Eingabe der Koerperlaenge in cm\n");
    scanf("%d",&laenge);
    ...
}
```

```
#include <stdio.h>
int main() { /* Eingaben und Ausgaben */
    int fusslaenge;
    printf("Eingabe der Fusslaenge in cm\n");
    scanf("%d",&fusslaenge);
    ...
}
```

# 4. Aufgabe Berechnung von Schuhgrößen

Vergleiche Wikipedia: Schuhgröße ([de.wikipedia.org/wiki/SchuhgröÙe](https://de.wikipedia.org/wiki/SchuhgröÙe))

...

Als **SchuhgröÙe** bezeichnet man eine alphanumerische Angabe für die GröÙe eines Schuhs oder Fußes. Meistens beschränkt sie sich auf eine numerische Angabe der Länge, da aus Kostengründen in Produktion oder bei der Bevorratung meist nur eine Einheitsweite angeboten wird. Es gibt mehrere verschiedene, nebeneinander existierende, historisch gewachsene SchuhgröÙensysteme, teils mit nationalen Schwerpunkten. Sie unterscheiden sich in der BezugsgröÙe, der Maßeinheit oder dem Skalen-Nullpunkt. Nur ein Teil der Systeme bezieht neben der Fußlänge auch die Fußbreite ein, sodass hierfür oft nicht standardisierte, herstelllerspezifische Bezeichnungen in Gebrauch sind.

*SchuhgröÙe (EU) = (Fußlänge in cm + 1,5) × 1,5*

*Brannock-System:*

*HerrengroÙe (US) = Fußlänge in cm ÷ 2,54 × 3 - 22*

*DamengroÙe (US) = Fußlänge in cm ÷ 2,54 × 3 - 21*

# 4. Aufgabe - Berechnung von Schuhgrößen

*Schuhgröße (EU) = (Fußlänge in cm + 1,5) × 1,5*

*Schuhgröße (in Deutschland gebräuchlich) = (Fußlänge in cm + 1,54) / 0.667*

*Brannock-System:*

*Herrengroße (US) = Fußlänge in cm ÷ 2,54 × 3 - 22*

*Damengroße (US) = Fußlänge in cm ÷ 2,54 × 3 - 21*

Schreiben Sie ein Programm schuh.exe, welches die benannten Blöcke schuhgroesse\_de(), schuhgroesse\_eu() und schuhgroesse\_usa() enthält und die Berechnung der Schuhgrößen vornimmt und die Ergebnisse ausgibt..

In der EU sind Schuhgrößen ganzzahlig, die Schuhgrößen im Brannock-System haben Abstufungen von 0.5. Sie können folgenden Programmrahmen nutzen:

```
#include <stdio.h>
int fusslaenge;
schuhgroesse_de() { // Hier Deutschland }
schuhgroesse_eu() { // Hier EU}
schuhgroesse_usa() { // Hier Brannock}

int main() { /* Eingaben und Ausgaben */
    printf("Eingabe der Fusslaenge in cm\n");
    scanf("%d",&fusslaenge);
    schuhgroesse_de();
    schuhgroesse_eu();
    schuhgroesse_usa();
}
```

## 4. Aufgabe - Berechnung von Schuhgrößen \*

Wie Runden auf 0.5 Abstufung?

Wie kann für Fußlängen von 26 cm bis 40 cm das Programm eine Tabelle ausgeben?

Berechnen Sie den Body-Mass-Index

Testen Sie ihr Wissen über RapidSVN!

# 4. Aufgabe BMI

Der Body-Mass-Index wird folgendermaßen berechnet:

$$\text{BMI} = m / l^2,$$

wobei  $m$  die Körpermasse (in Kilogramm) und  $l$  die Körpergröße (in m) angibt.

Eingabe von Größe in cm, Masse kg!