

Variablen und Datentypen

Einführung in die Programmierung

Michael Felderer (QE)

Institut für Informatik, Universität Innsbruck

Variablen

- C ist eine **imperative** Programmiersprache
 - Programm ist eine Abfolge von Anweisungen die auf Speicherstellen operieren
- Variable
 - Ein Programm verarbeitet **Daten**, die in sogenannten **Variablen** abgelegt werden.
 - Ein Programm legt die **Ergebnisse** wieder in solchen **Variablen** ab.
 - Eine Variable ist eine **benannte Speicherstelle**.
- Kennzeichen (für eine Variable)
 - Name
 - Datentyp
 - Wert
 - Adresse
 - Gültigkeitszeitraum
 - Sichtbarkeitsbereich

Kennzeichen (1)

- Name (Bezeichner)
 - Damit greift der Programmierer auf die benannte Speicherstelle zu.
- Datentyp
 - Daten haben einen bestimmten Typ (Datentyp).
 - Der Typ schränkt die Benutzung der Daten ein.
 - Der Datentyp legt fest
 - **Wertebereich**,
 - **Repräsentation im Speicher** (wie viele Bytes werden belegt),
 - **Operationen**, die auf den Datentyp angewandt werden dürfen.
- Wert
 - Der Wert einer Variable kann sich ändern.
 - Der Variable muss in bestimmten Situationen durch das Programm am Anfang explizit ein Wert zugewiesen werden (**Initialisierung**).
 - Ansonsten hat die Variable einen zufälligen Wert!
 - **Man kann dabei nicht davon ausgehen, dass die Speicherstelle mit 0 belegt ist!**

Kennzeichen (2)

- Adresse
 - Die Variablen liegen in den Speicherzellen des Arbeitsspeichers.
 - Die Speicherzellen des Arbeitsspeichers sind durchnummeriert und diese Nummer bezeichnet man als Adresse.
 - Eine Variable kann (und wird auch sehr oft) eine Folge zusammenhängender Speicherzellen belegen.
 - Die Adresse der Variable ist dann die Nummer der Speicherzelle, in der die Variable beginnt.
- Gültigkeitszeitraum und Sichtbarkeit werden später genauer besprochen!

Beispiel (Variablen definieren)

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

```
int main(void) {
    short counter;
    char letter;
    int number1;
```

Hier werden drei Variablen definiert.
Zuerst wird der Datentyp, dann der
Bezeichner angegeben. Erklärungen dazu
folgen auf den nächsten Folien.

...

```
    return EXIT_SUCCESS;
}
```

- **Definition** einer Variablen = Deklaration + Reservierung des Speicherplatzes
- **Deklaration** umfasst Namen und Typ einer Variablen und macht dem Compiler bekannt mit welchem Typ ein bestimmter Variablenname verbunden werden muss

Namen

- Name (Bezeichner) einer Variable
 - Ist eine Folge aus Buchstaben, Ziffern und Unterstrich.
 - Der Name muss aber mit Buchstabe oder Unterstrich beginnen.
 - Die Variablennamen werden stets klein geschrieben (Konvention).
 - Der Unterstrich am Anfang sollte vermieden werden (wird nur in Bibliotheken verwendet).
 - C ist „case-sensitive“, d.h. in C werden Klein- und Großbuchstaben unterschieden!

<i>Korreakter Name</i>	<i>Nicht korrekter Name</i>
counter	1counter
carwheel	car-Wheel oder car/Wheel
max_count	while
test2	\$%

Sinnvolle Namen

- Die Namen sollten bestimmten Regeln folgen!
 - Sie sollten lesbar und verständlich sein.
 - Compiler wird alle korrekten Namen akzeptieren.
 - Ein korrekter Name muss aber noch lange nicht sinnvoll sein!

	<i>Negatives Beispiel</i>	<i>Positives Beispiel</i>
<i>Vollständige Wörter</i>	c (siehe Hinweis)	counter
<i>Kleinschreibung</i>	COUNTER (siehe Hinweis)	counter
<i>Neue Teile mit Unterstrich beginnen</i>	firstletterintext	first_letter_in_text
<i>Sinnvolle Bezeichner</i>	\$0x01	counter
<i>Abkürzungen vermeiden</i>	bc	byte_count
<i>Englische Bezeichner verwenden</i>	erschta_buchstobn	first_letter

- Hinweis
 - Einzelne Buchstaben (wie c) sind manchmal sinnvoll bzw. werden in der Praxis verwendet (z.B. Variablen für Schleifen).
 - Großschreibung (wie COUNTER) wird bei Konstanten verwendet.

Reservierte Schlüsselwörter in ANSI C/C99

- Werden immer klein geschrieben.
- Können **nicht** als Variablennamen verwendet werden.

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				
restrict	inline	_Bool	_Complex	_Imaginary	

- Die Schlüsselwörter in der letzten Zeile (rot geschrieben) existieren erst in C99.

Datentypen

- In einem Computer werden Zeichen (z. B. Buchstaben) anders behandelt als ganze Zahlen und Gleitkommazahlen wie z.B. die Zahl $\pi = 3.1415...$
 - Daher ist eine Klassifikation dieser unterschiedlichen Daten notwendig.
 - Ordnet man in einem Programm Daten bestimmten Klassen wie Zeichen, ganze Zahl, einfach/doppelt genaue Gleitkommazahl usw. zu, dann teilt man dem Rechner deren Datentyp mit.
- In C hat jede Variable einen genau definierten, vom Programmierer festgelegten Typ.
 - Der Typ bestimmt, welche Werte eine Variable annehmen kann und welche nicht.
 - Der Datentyp einer Variable legt die Darstellung durch den Rechner fest:
 - Speicherbedarf (durch wie viele Bits wird die Variable dargestellt ?)
 - Wertebereich und Zeichensatz
 - Genauigkeit (bei Gleitkommazahlen)

Elementare (einfache) Datentypen

- **char**
 - Dient zur Aufnahme von Zeichen (oder Ganzzahlen).
- **int**
 - Wird für ganzzahlige Werte mit Vorzeichen verwendet.
- **float**
 - Wird für Gleitkommazahlen mit einfacher Genauigkeit verwendet.
- **double**
 - Wird für Gleitkommazahlen mit doppelter Genauigkeit verwendet.

Typmodifikatoren

- Die elementaren Datentypen können durch Typmodifikatoren angepasst werden.
- **short int:**
 - Ganzzahliger Datentyp, der höchstens den Wertebereich von **int** besitzt; **int** kann weggelassen werden.
- **long int:**
 - Ganzzahliger Datentyp, der mindestens den Wertebereich von **int** besitzt; **int** kann weggelassen werden.
- **long double:**
 - Gleitkomma-Datentyp mit erweiterter Genauigkeit; kann je nach System identisch mit **double** sein.
- **long long int:**
 - Erweiterter ganzzahliger Typ; kann identisch mit **long** sein; **int** kann weggelassen werden.

signed oder unsigned

- **char** und die ganzzahligen Datentypen können auch hinsichtlich der Berücksichtigung des Vorzeichens modifiziert werden.
- Dazu wird der jeweiligen Datentypbezeichnung einer der folgenden zwei Modifikatoren vorangestellt:
 - **unsigned**: Das Vorzeichenbit wird für die Darstellung des positiven Zahlenwerts frei.
 - **signed**: Ein Bit wird für die Darstellung des Vorzeichens reserviert.
- Standardmäßig wird bei Ganzzahlen immer **signed** benutzt.

Beispiele für Datentypen (zid-gpl)

Datentyp (alternative Bezeichnungen)	Bitanzahl 32 Bit (64 Bit)	Wertebereich
char (signed char)	8	-128...127
unsigned char	8	0...255
short (signed short, short int, signed short int)	16	-32768...32767
unsigned short (unsigned short int)	16	0...65535
int , signed, signed int	32	-2147483648...2147483647
unsigned (unsigned int)	32	0...4294967295
long (signed long, long int, signed long int)	32 (64)	-2147483648...2147483647 (siehe long long)
unsigned long (unsigned long int)	32 (64)	0...4294967295 (siehe long long)
long long (signed long long, long long int, signed long long int)	64	-9223372036854775808... 9223372036854775807
unsigned long long (unsigned long long int)	64	0...18446744073709551615
float	32	$1.2 \cdot 10^{-38} \dots 3.4 \cdot 10^{38}$
double	64	$2.2 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$
long double	128	$3.4 \cdot 10^{-4932} \dots 1.2 \cdot 10^{4932}$

Größe der Datentypen

- Der C-Standard gibt minimale Größen an.
 - Diese sollten aber durch entsprechende Werte ersetzt werden.
 - Die Größe ist daher **immer abhängig vom System!**
- Größen werden in den folgenden Headerdateien genau festgelegt:
 - `limits.h`
 - `float.h`
 - `stdint.h` (Integer mit bestimmter Größe)

Variablen vereinbaren

- Vor der ersten Verwendung muss eine Variable vereinbart werden:
 - **Datentyp**
 - **Name**
- Beispiele
 - **short counter;**
 - Datentyp **short**, d.h. mindestens 2 Bytes werden für eine Ganzzahl im 2-er Komplement reserviert (am zid-gpl!).
 - Der Name der Variable ist **counter**.
 - **char letter1, letter2;**
 - 2 Variablen **letter1** und **letter2**.
 - Jeweils 1 Byte wird für jede Variable verwendet (am zid-gpl!).
 - Der Inhalt wird als Zeichen interpretiert (ASCII).
 - **double north, northeast, east, southeast, south, southwest, west, northwest;**
 - 8 Variablen vom Typ double (8 Bytes am zid-gpl).
 - Sind alles Gleitkommazahlen mit doppelter Genauigkeit.

Variablen vereinbaren (falsche Beispiele)

- **short counter1; counter2;**
 - Zweite Variable ohne Datentyp (; schließt Vereinbarung ab).
- **int first, int second;**
 - Datentyp bei mehreren Variablen nur einmal angeben.
- **float a, double b;**
 - Wie vorher (auch mit unterschiedliche Datentypen nicht möglich).
- **first_value int;**
 - Name steht an erster Stelle und wird daher als Datentyp angesehen, dieser Datentyp existiert aber ursprünglich nicht in C.
 - Bezeichner ist ein reserviertes Schlüsselwort.

Interaktive Aufgabe

- Welche der folgenden Anweisungen sind gültig?

`unsigned short int a = -1;`

`int b = 7; double c;`

`int d, int e;`

`long int f, g;`

`double h; i=2;`

`j float;`

Beispiel (Variablen)

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

int main(void) {
    short counter = 2;
    char letter1 = 'a', letter2 = 'b';
    int number1 = 100000;
    double number2 = 10.0, number3 = 20;

    printf("%hd\n", counter);
    printf("%c, %c\n", letter1, letter2);
    printf("%d, %f, %f\n", number1, number2, number3);

    return EXIT_SUCCESS;
}
```

Ausgabe:

2

a, b

100000, 10.000000, 20.000000

Formatierung:

%d – eine Zahl ausgeben (für short zusätzlich h)

%c – ein Zeichen ausgeben

%f – eine Gleitkommazahl (double) ausgeben

\n – in eine neue Zeile springen

Variablen initialisieren

- Warum wird in dem Beispiel immer ein Wert zugewiesen?
- Variablen, die auf diese Art und Weise in der main-Funktion definiert werden, haben keinen sinnvollen Wert!
- **Daher muss jede Variable initialisiert werden!**

Datentyp char – Spezialfall

- Dieser Datentyp kann auf zwei völlig unterschiedliche Arten genutzt werden.

- Darstellung von Zeichen
- Darstellung von kleinen Ganzzahlen

- Zuweisung eines Zeichens

```
char ch = 'A';
```

- Achtung: Einfache Hochkommata benutzen!

- Zuweisung einer Zahl

```
char ch = 65;
```

- Interpretation

```
printf("%c %hhd\n", ch, ch); // Ausgabe: A 65
```

Datentyp char – Zeichensatz

- Achtung
 - Umwandlung von Zahlenwerten zu Buchstaben hängt vom verwendeten Zeichensatz ab (obiges Beispiel geht vom ASCII-Zeichensatz aus)!
 - Man kann nicht davon ausgehen, dass zwei aufeinanderfolgende Zeichen auch aufeinanderfolgende Ordinalwerte (Zahl, die dem Zeichen entspricht) haben.
- Der C-Standard legt nicht fest, welcher Zeichensatz verwendet wird.

Zeichensatz	Ordinalwert	Speicherbedarf	Bemerkung
ASCII	0 bis 127	7 Bit	
OEM	0 bis 255	8 Bit	Mit ASCII-Code
ANSI	0 bis 255	8 Bit	Mit ASCII-Code
ISO-Latin-1	0 bis 255	8 Bit	Mit ASCII-Code
Unicode	0 bis 65535	16 Bit	Mit ASCII-Code
EBCDIC	0 bis 255	8 Bit	

Interaktive Aufgabe

- Im folgenden Codeausschnitt wird jeweils einer char-Variable ein Zeichen übergeben. Welche Zuweisungen sind falsch?

```
char ch1 = ' ';
```

```
char ch2 = 66;
```

```
char ch3 = "B";
```

```
char ch4 = '\x42';
```

```
char ch5 = ch1;
```

```
char ch6 = 0x42;
```

```
char ch7 = B;
```

Boolescher Datentyp

- Im C99-Standard wurde mit `_Bool` ein boolescher Datentyp eingeführt.
- Wenn man `<stdbool.h>` inkludiert
 - Dann kann man auch `bool` schreiben.
 - Dann sind die Werte `true` und `false` (via Makros) vorhanden.
 - Beispiel

```
_Bool a = true;
bool b = false;
```
- Anmerkung
 - Wird aktuell noch selten verwendet.
 - Sinnvoll ist dieser Datentyp bei der Überprüfung von logischen Ausdrücken.
 - Logische Ausdrücke werden im nächsten Foliensatz erklärt.

Definition und Deklaration

- Eine Vereinbarung umfasst sowohl eine Definition als auch eine Deklaration
 - Definitionen
 - Legen die Art der Variablen fest und sorgen **gleichzeitig** dafür, dass zur Laufzeit Speicherplatz reserviert wird.
 - Deklarationen
 - Legen nur die Art der Variablen fest.
 - Eine bestimmte Deklaration kann auch mehrmals in einem C-Programm vorkommen (Beispiele folgen noch).
 - Dienen dazu, Datenobjekte bekannt zu machen.
- Ort der Definition
 - In ANSI C muss eine Definition am Anfang eines Blocks stattfinden.
 - Blocksemantik wird noch besprochen.
 - Ein Beispiel dafür ist der Anfang der **main**-Funktion.
 - In C99 kann eine Variable auch in einer beliebigen Zeile eines Blocks definiert werden.

Ganzzahlige Literale

- Literale
 - Zahlen, Zeichenketten und Wahrheitswerte im Quelltext
- Beispiele
 - **1234** = ganzzahliges Literal vom Typ **int**.
 - **1234L** = ganzzahliges Literal vom Typ **long**.
 - **1000000000000** = ganzzahliges Literal vom Typ **long** (zu groß für **int**).
 - **0XFULL** = hexadezimaler Literal **unsigned long long** 15.
- Drei Arten
 - Dezimal: Beginnt mit einer Zahl von 0 verschieden.
 - Oktal: Beginnt immer mit 0.
 - Hexadezimal: Beginnt immer mit 0x.
- Suffix am Ende anhängen
 - **u** oder **U** (**unsigned**), **l** oder **L** (**long**), **ul**, **uL**, **Ul**, **UL** (**unsigned long**), **ll** oder **LL** (**long long**), **ull**, **uLL**, **Ull**, **ULL** (**unsigned long long**)

Gleitkommalliterale

- Gleitkommalliterale enthalten einen Dezimalpunkt oder einen Exponenten oder beides.
- Suffix
 - Ohne: double
 - f,F: float
 - l,L: long double
- Beispiele
 - **10.5** = Gleitkommaliteral vom Typ double.
 - **10.5f** = Gleitkommaliteral vom Typ float.
 - **19e-2** = Gleitkommaliteral vom Typ double (Exponentenschreibweise).
- Verschiedene Darstellungen sind möglich (z.B. für 0.19):
 - **0.19**
 - **19e-2** oder **1.9e-1**
 - **.19**

Zeichenliteral

- Ein Zeichenliteral ist ganzzahlig und wird als Einzelzeichen innerhalb von **einfachen** Anführungszeichen geschrieben.
 - 'a', 'W' etc.
- Der Wert eines Zeichenliterals entspricht dem numerischen Wert des Zeichens im Zeichensatz der Maschine.
 - '0' hat zum Beispiel im ASCII-Zeichensatz den Wert 48.
- Ersatzdarstellung
 - Wird verwendet für Steuerzeichen und spezielle Zeichen.
 - Es ist immer nur ein Zeichen damit gemeint.
 - Beispiele:
 - \n Zeilentrenner, \t Tabulatorzeichen ...
 - \' einfaches Hochkomma
 - \000 oktale Zahl, \xhh hexadezimale Zahl

Ausgabe mit printf (1)

- `printf`
 - gehört nicht zur Sprache C selbst, sondern
 - ist eine Funktion der C Standard Library.
- Allgemeine Form von `printf`
 - `printf(Formatstring, Parameterliste)`
- Parameterliste
 - Die Parameterliste hängt vom Formatstring ab und fällt ggf. ganz weg.
 - Die Parameterliste kann Konstanten, Variablen oder Ausdrücke enthalten.
- Formatstring
 - Besteht aus gewöhnlichem Text, in dem man Platzhalter zur Ausgabe von Variableninhalten einfügen kann.
 - Die Platzhalter beginnen immer mit einem Prozentzeichen %.
 - Platzhalter können auch Angaben über das Ausgabeformat enthalten.

Ausgabe mit printf (2)

Zeichen	Bedeutung
d,i	Ausgabe als ganze (positive oder negative) Dezimalzahl (Datentyp <code>int</code>).
u	Ausgabe als ganze positive Dezimalzahl (Datentyp <code>unsigned int</code>).
o	Ausgabe als ganze positive Oktalzahl (Datentyp <code>unsigned int</code>).
x, X	Ausgabe als ganze positive Hexadezimalzahl in Klein- bzw. Großschreibung (Datentyp <code>unsigned int</code>).
c	Ausgabe eines einzelnen Zeichens (Datentyp <code>int</code>).
s	Ausgabe einer Zeichenkette (wird noch ausführlich besprochen).
f	Ausgabe als Fließkommazahl (Datentyp <code>double</code>).
e, E	Ausgabe mit Exponentialdarstellung mit kleingeschriebenem „e“ bzw. großgeschriebenem „E“.
g, G	Ausgabe als Exponentialzahl bzw. als Dezimalzahl in Abhängigkeit vom Wert.
a, A (C 99)	Ausgabe als Exponentialzahl in hexadezimaler Darstellung.
p	Ausgabe als Adresse (Ausgabe ist implementierungsabhängig).
n	Speichert in einer als Argument angegebenen Variable des Typs <code>int</code> die Anzahl der bisher ausgegebenen Zeichen.
%	Ausgabe des %-Zeichens.

Ausgabe mit printf (3)

- Platzhalter im Formatstring haben die folgende Struktur:
 - **%[flags][weite][.genauigkeit][modifizierer]typ**
 - Angaben in Klammern sind optional.
 - **typ**
 - Spezifiziert den Umwandlungsspezifikator (siehe Tabelle auf vorheriger Folie).
 - **modifizierer**
 - Bezieht sich auf den Wert von typ (siehe Tabelle auf der nachfolgenden Folie).
 - **genauigkeit**
 - Gibt die Anzahl der Nachkommastellen bzw. Stellen für Ganzzahlen an; wenn weniger Stellen vorhanden sind, dann wird mit Nullen aufgefüllt; wenn nicht vorhanden, dann wird bei float/double mit 6 Nachkommastellen gearbeitet.
 - **weite**
 - Gibt die Mindestanzahl der Zeichen bei der Ausgabe an. Ist die Ausgabe kürzer, so wird mit Leerzeichen aufgefüllt, ist sie länger, dann wird dieser Wert ignoriert.
 - **flags**
 - Hier kommen zum Beispiel in Frage
 - – (linksbündige Ausgabe) oder + (positive Zahl mit Vorzeichen ausgeben)
 - 0 (mit führenden Nullen auffüllen), # (alternative Form – siehe Doku)
 - Leerzeichen (wird vorangestellt, wenn das erste Zeichen kein Vorzeichen ist)

Ausgabe mit printf (4)

Zeichen	Bedeutung
h	Ganzzahlumwandlung als short (signed, unsigned)
hh	Ganzzahlumwandlung entspricht char (signed, unsigned)
j	Ganzzahlumwandlung entspricht einem Argument vom Typ <code>intmax_t</code> oder <code>uintmax_t</code> .
l	Ganzzahlumwandlung als long (signed, unsigned)
ll	Ganzzahlumwandlung als long long (signed, unsigned)
L	Eine folgende a-, A-, e-, E-, f-, F-, g- oder G-Umwandlung entspricht einem long double-Argument
t	Ganzzahlumwandlung entspricht einem Argument vom Typ <code>ptrdiff_t</code> (Differenz zweier Zeiger)
z	Ganzzahlumwandlung entspricht einem Argument vom Typ <code>size_t</code> oder <code>ssize_t</code> .

Beispiel (Literele)

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

int main(void) {
    int a = 123;
    long b = 12341;
    //int c = 10000000000; Overflow!
    unsigned long long d = 0x12FULL;
    float e = 10.0f;
    double f = .0125;
    char g = 'a';
    printf("%d\n", a);
    printf("%ld\n", b);
    printf("%lld\n", d);
    printf("%f\n", e);           // default argument promotion
    printf("%f\n", f);
    printf("%c\n", g);
    printf("%hhd\n", g);
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
123
1234
303
10.000000
0.012500
a
97
```


Interaktive Aufgabe

- Was wird durch das folgende C-Programm ausgegeben?

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

int main(void) {

    int a = 0xF;
    int b = 16;
    unsigned short c = 65535;
    float d = 10e-2;
    double e = .25;

    printf("%d\n", a);
    printf("%x\n", b);
    printf("%hu\n", c);
    printf("%f\n", d);
    printf("%e\n", e);
    return EXIT_SUCCESS;
}
```



- Wie merkt man sich all diese Umwandlungsspezifikatoren und anderen Details?
 - Man schlägt sie im Handbuch nach!
 - Auf entsprechend eingerichteten UNIX-Rechnern:
 - `man 3 printf`
 - Lernen Sie möglichst bald, UNIX-Manpages zu lesen!

Aufzählungskonstanten

- Eine Aufzählung ist eine Folge von konstanten ganzzahligen Werten.
`enum boolean {FALSE, TRUE};`
 - Typ **boolean** mit zwei Aufzählungskonstanten.
 - Eine Variable von Typ **boolean** kann einen Wahrheitswert repräsentieren.
 - Hinweis: In C99 gibt es dafür einen eigenen Typ!
- Aufzählungskonstanten haben einen konstanten ganzzahligen Wert (vom Typ **int**).
 - Die erste Konstante hat den Wert 0, die zweite den Wert 1 usw.
 - Werte können auch mit ganzen Zahlen beliebig belegt werden.

Beispiel (enum)

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

int main(void) {
    enum boolean {FALSE, TRUE} bool;
    enum test1 {ALPHA = 5, BETA = 3, GAMMA = 6};
    enum months {JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV,
DEZ};
    bool = TRUE;
    enum test1 t1 = GAMMA;
    enum months mon = APR;
    enum test1 t2 = 100;

    printf("%d\n", bool);
    printf("%d\n", t1);
    printf("%d\n", mon);
    printf("%d\n", t2);

    return EXIT_SUCCESS;
}
```

Nicht sinnvolle
Zuweisung wird vom
Compiler nicht
überprüft!

Ausgabe:

1
6
4
100

(Symbolische) Konstanten

- Mit der **define**-Direktive (am Anfang des Programms):

```
#define PI 3.141592654
```

```
...
```

```
x = y * PI;
```

- Definition unter Verwendung des Schlüsselwortes **const**:

```
const double e = 2.71828182;
```

```
...
```

```
x = e * 2;
```

- Unterschied
 - **define**-Direktive wird nur textuell vom Präprozessor ersetzt (keine Typprüfung vom Compiler!)
- Beispiel für solch eine Konstante ist **EXIT_SUCCESS**.
 - Der eigentliche Wert hängt vom System ab (meistens 0).
 - Wird die Konstante verwendet, dann wird sich das übersetzte Programm auf beliebigen Betriebssystemen korrekt verhalten.

Beispiel (printf – teilweise falsche Formatierung)

```
#include <stdio.h>
#include <stdlib.h>
#define PI 3.141592654

int main(void) {
    int a = 40, b = -30;
    char c = 'B';
    long d = 1000000000;
    float e = 123.456789123;
    printf("%d\n", a);
    printf("%d\n", b);
    printf("%u\n", a);
    printf("%u\n", b);
    printf("%x\n", a);
    printf("%c\n", c);
    printf("%hhd\n", c);
    printf("%ld\n", d);
    printf("%f\n", e);
    printf("%e\n", e);
    printf("%g\n", e);
    printf("%f\n", PI);
    printf("%10d\n", a);
    printf("%10.2f\n", e);
    printf("%010.2f\n", e);
    printf("%+010.2f\n", e);
    printf("%+030.20f\n", e);
    return EXIT_SUCCESS;
}
```

Ausgabe:

```
40
-30
40
4294967266
28
B
66
1000000000
123.456787
1.234568e+02
123.457
3.141593
          40
        123.46
0000123.46
+000123.46
+00000123.45678710937500000000
```

↑
Ungenauigkeit bei IEEE 754

Interaktive Aufgabe

- Was wird durch das folgende C-Programm ausgegeben?

```
#include <stdio.h>
#include <stdlib.h>
#define E 2.718281828459045

int main(void) {
    int a = 9;
    short b = -2;
    char c = 'A'; // entspricht 65
    float d = 1234.5678;

    printf("%o\n", a);
    printf("%hu\n", b); // 2^16 = 65536
    printf("%d\n", b);
    printf("%c\n", c);
    printf("%d\n", c);
    printf("%f\n", d);
    printf("%e\n", d);
    printf("%g\n", d);
    printf("%f\n", E);
    printf("%5.3d\n", a);
    printf("%5.3f\n", d);
    printf("%010.3f\n", d);
    return EXIT_SUCCESS;
}
```

getchar und putchar

- 2 nützliche Funktionen (für die Aufgaben):
 - **getchar** liest ein einzelnes Zeichen von der Tastatur.
 - Liefert einen Integer zurück.
 - **putchar** gibt ein Zeichen auf dem Bildschirm (Konsole) aus.
 - **putchar** wird ein Integer übergeben.
- Die Funktionsweise und weitere Funktionen werden noch in späteren Kapiteln genauer besprochen.

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

```
int main() {  
    int c;  
  
    while ((c = getchar()) != EOF)  
        putchar(c);  
    return 0;  
}
```


scanf

- Die Funktion **scanf** liest zeichenweise eine Folge von Eingabefeldern ein.
 - Für jedes Eingabefeld muss eine Adresse vorhanden sein, wobei das Eingabefeld mit dem Datentyp der Adresse übereinstimmen muss.
 - Von einer Variable x kann man die Adresse mit &x ermitteln.
- Die Formatierung ist ähnlich zu **printf**.
 - **scanf** liest aber ein!
 - Manual-Seite studieren (man scanf)!
- Einfaches Beispiel (Wert in eine Integer-Variable einlesen):

```
int i;  
printf("Bitte geben Sie eine Zahl ein: ");  
scanf("%d", &i);  
printf("Die Zahl, die Sie eingegeben haben, war %d\n", i);
```
- Die genauen Details und Probleme werden in der Vorlesung über Zeiger noch ausführlich behandelt!

Interaktive Aufgabe

- Finden und beheben Sie die Fehler im folgenden Listing!

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

int main(void) {

    int var1, var2;
    printf("Zahl 1 eingeben:");
    scanf("%d", var1);
    printf("Zahl 2 eingeben:");
    scanf("%c", var2);
    printf("%d + %d = %d\n", var1+var2);
    return EXIT_SUCCESS;
}
```