

## Kurseinheit 2: Operatoren sowie formatierte Ein- und Ausgabe

1. Operatoren
2. Formatierte Ausgabe
3. Formatierte Eingabe

## Übersicht KE 2

Lehrveranstaltung Ingenieur-Informatik – 2 SWS/2 Credits: EI1, EI+1, MKA1, MK+1, EI3nat3  
Lehrveranstaltung Programmierung 2 (Teil C) – 2 SWS/2 Credits: AI2

Unterrichtsdauer für diese Kurseinheit: 90 Minuten

Korrespondierende Kapitel aus *C-Programmierung – Eine Einführung*: Kapitel 3 und 5

Zusatzthemen: Keine

## 1. Operatoren

3

### Klassifikation Operatoren

Klassifikation nach der Anzahl der Operanden:

- Operator hat nur einen Operanden: **unärer** Operator: `-iVal1`
- Operator hat zwei Operanden: **binärer** Operator: `iVal1 + iVal2`
- Operator hat drei Operanden: **ternärer** Operator:  
`(iVal1 < 10) ? (iVal2 = 1) : (iVal2 = 73)`

Operatorsymbole + und - können für unäre und binäre Operatoren verwendet werden.  
Leerzeichen siehe CL5 C-Coding Styleguide.

Klassifikation nach den Funktionsbereichen der Operatoren:

- Arithmetische Operatoren
- Relationale Operatoren
- Logische Operatoren
- Bitoperatoren
- Zuweisungsoperatoren
- Umwandlungsoperatoren (Casts)

Operatoren haben unterschiedliche **Prioritäten** und **Assoziativitäten**. Siehe hierzu PA1 und zusätzlich noch PA2, PA3 und PA4 des C-Coding Styleguides. Dies ist besonders wichtig, wenn verschiedene Operatoren in einer C-Anweisung verwendet werden.

## 1. Operatoren

4

### Arithmetische Operatoren

Operator	Bedeutung	Beispiel(e)
-	Subtraktion und unäres Minus	<code>iY = iX - iZ; iY = iX - 73; iY = -iX;</code>
+	Addition (und unäres Plus)	<code>iY = iX + iZ; iY = iX + 73; iY = +iX;</code>
*	Multiplikation	<code>iY = iX * iZ; iY = iX * 73; iY = 73 * iX;</code>
/	Division	<code>iY = iX / iZ; iY = iX / 73;</code> <code>iY = 73 / iX; iY = 73 / 74;</code>
%	Modulo Division	<code>iY = iX % 2;</code>
--	Dekrement – PreDekrement Dekrement – PostDekrement	<code>iY = --iX; --iY; (iY = iY - 1;)</code> <code>iY = iX--; iY--; (iY = iY - 1;)</code>
++	Inkrement – PreInkrement Inkrement – PostInkrement	<code>iY = ++iX; ++iY; (iY = iY + 1;)</code> <code>iY = iX++; iY++; (iY = iY + 1;)</code>

**Division durch Null** ergibt einen Laufzeitfehler (Programmabbruch). Bei einer Ganzzahldivision wird der Rest abgeschnitten. `73 / 74` ergibt 0!

**Modulo Division %** ist in der Informatik ein ganz wichtiger Operator (ganzzahliger Rest)!

## 1. Operatoren

5

### Arithmetische Operatoren - Codebeispiel

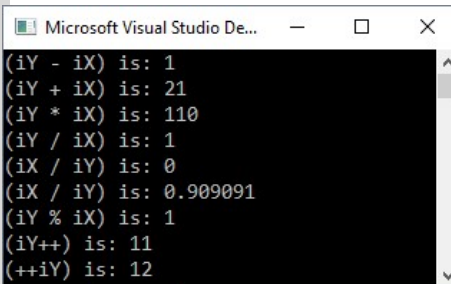
```
int iY = 11; ←
int iX = 10; ←
float fY = 11.0F; ←
float fX = 10.0F; ←
int iRes;
// long version
iRes = iY - iX;
printf("(iY - iX) is: %d\n", iRes);
// short version
printf("(iY + iX) is: %d\n", (iY + iX));
printf("(iY * iX) is: %d\n", (iY * iX));
printf("(iY / iX) is: %d\n", (iY / iX));
printf("(iX / iY) is: %d\n", (iX / iY));
printf("(fX / fY) is: %f\n", (fX / fY));
printf("(iY %% iX) is: %d\n", (iY % iX));

iY = 11;
printf("(iY++) is: %d\n", (iY++));
iY = 11;
printf("(++iY) is: %d\n", (++iY));
```

Literale Integer-Konstanten

Literale Gleitkomma-Konstanten

Bei der long version wird der Wert erst in einer Hilfsvariable (iRes) gespeichert und diese dann ausgegeben.



```
(iY - iX) is: 1
(iY + iX) is: 21
(iY * iX) is: 110
(iY / iX) is: 1
(iX / iY) is: 0
(iX / iY) is: 0.909091
(iY % iX) is: 1
(iY++) is: 11
(++iY) is: 12
```

## 1. Operatoren

6

### Relationale Operatoren

Operator	Bedeutung	Beispiel(e)	
>	Größer	iY > 10	iY > iX
>=	Größer als oder gleich	iY >= 10	iY >= iX
<	Kleiner	iY < 10	iY < iX
<=	Kleiner als oder gleich	iY <= 10	iY <= iX
==	Gleich	iY == 10	iY == iX
!=	Ungleich	iY != 10	iY != iX

Relationale Operatoren liefern die Wahrheitswerte true (1) und false (0) zurück.

In C89 wird ein Wert ungleich 0 als wahr angesehen. Die relationalen (und später auch die logischen) Operatoren liefern aber immer den Wert 1 als Rückgabewert für true.

## 1. Operatoren

7

### Relationale Operatoren - Codebeispiel

```
int iY = 11;
int iX = 10;
int iRes;

iRes = iY < iX; // long version
printf("Result (iY < iX) is: %d\n", iRes); // long version
printf("Result (iY < iX) is: %d\n", (iY < iX)); // short version
printf("Result (iY >= iX) is: %d\n", (iY >= iX)); // short version
```

## 1. Operatoren

8

### Logische Operatoren

Operator	Bedeutung	Beispiel(e)
&&	Logisches UND	(iY > 10) && (iX < iZ)
	Logisches ODER	(iY > 10)    (iX < iZ)
!	Logisches NOT	!(iY) <b>Kein Leerzeichen nach !</b>

Logische Operatoren liefern die Wahrheitswerte true (1) und false (0) zurück. Diese werden meist mit relationalen Operatoren kombiniert.

#### Wahrheitstabellen

A	B	A && B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A    B
0	0	0
0	1	1
1	0	1
1	1	1

A	!A
0	1
1	0

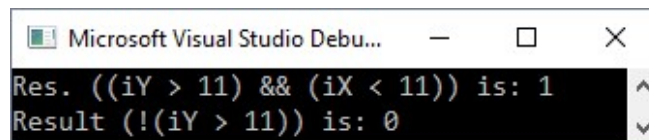
## 1. Operatoren

9

### Logische Operatoren - Codebeispiel

```
int iY = 12;
int iX = 10;

printf("Res. ((iY > 11) && (iX < 11)) is: %d\n", ((iY > 11) && (iX < 11)));
printf("Result (!(iY > 11)) is: %d\n", (!(iY > 11)));
```



```
Microsoft Visual Studio Debu...
Res. ((iY > 11) && (iX < 11)) is: 1
Result (!(iY > 11)) is: 0
```

## 1. Operatoren

10

### Bitoperatoren

Operator	Bedeutung	Beispiel(e)
&	Bitweises AND	ucC = ucA&ucB; ucC = ucA&0x0F;
	Bitweises OR	ucC = ucA ucB; ucC = ucA 0x0F;
^	Bitweises EXOR	ucC = ucA^ucB; ucC = ucA^0x0F;
~	Bitweises NOT	ucC = ~ucA; ucC = ~0x0F;
>>	Bitschieben rechts	ucC = ucA>>2U; ucC = ucA>>ucB;
<<	Bitschieben links	ucC = ucA<<2U; ucC = ucA<<ucB;

**Empfehlung:** Bitoperatoren immer nur auf unsigned Ganzzahlen-Variablen anwenden. Hier gibt es auch ein „Exklusiv-ODER“ (EXOR). **Keine Leerzeichen vor und nach dem Operator.**

**Wahrheitstabelle für bitweises EXOR**

Bit1	Bit2	Bit1^Bit2
0	0	0
0	1	1
1	0	1
1	1	0

Unterschied zu  
bitweise OR

# 1. Operatoren

11

## Bitoperatoren - Basics

Dezimal	Hexa-dezimal	Binär
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Ein Byte (8 Bit) kann mittels zweier Hexadezimalziffern dargestellt werden ( $2 \cdot 4$  Bit). Tabelle links wird auch in der Klausur im Anhang ausgeteilt. Um die unterschiedlichen Zahlensysteme zu erkennen, wird häufig die Basis rechts tiefergestellt angegeben oder bei Hexadezimalzahlen ein 0x vorangestellt.

### Beispiele:

Zahlen im Bereich 0-15:

$$11_{10} = B_{16} = 0xB = 1011_2$$

Zahlen im Bereich 16-255:

$$16_{10} = 10_{16} = 0x10 = 00010000_2 = 10000_2$$

$$237_{10} = ED_{16} = 0xED = 11101101_2$$

$$255_{10} = FF_{16} = 0xFF = 11111111_2$$

Hinweis: Calc (Taschenrechner unter Windows):

->Ansicht -> Programmierer

Hex, Dez, Okt und Bin als Zahlenformate

# 1. Operatoren

12

## Bitoperatoren - Codebeispiel

```
unsigned char ucChar1 = 0xF0;
unsigned char ucChar2 = 0xA5;
printf("Result (ucChar1&ucChar2): %X\n", (ucChar1&ucChar2));
printf("Result (ucChar1|ucChar2): %X\n", (ucChar1|ucChar2));
printf("Result (ucChar1^ucChar2): %X\n", (ucChar1^ucChar2));
printf("Result (~ucChar1): %X\n", (~ucChar1));

ucChar1 = 0x18;
ucChar2 = 2U; // 0x02
printf("Result (ucChar1<<ucChar2): %X\n", (ucChar1<<ucChar2));
printf("Result (ucChar1>>ucChar2): %X\n", (ucChar1>>ucChar2));

//Last Operand is literal constant
ucChar1 = 0xF0;
ucChar2 = 0xA5;
printf("Result (ucChar1&0xA5): %X\n",
      (ucChar1&0xA5));
printf("Result (ucChar1>>2U): %X\n",
      (ucChar1>>2U));
```

```
Microsoft Visual ...
Result (ucChar1&ucChar2): A0
Result (ucChar1|ucChar2): F5
Result (ucChar1^ucChar2): 55
Result (~ucChar1): FFFFFFFF
Result (ucChar1<<ucChar2): 60
Result (ucChar1>>ucChar2): 6
Result (ucChar1&0xA5): A0
Result (ucChar1>>2U): 3C
```

# 1. Operatoren

13

## Zuweisungsoperator - Kurzformen

Der Zuweisungsoperator = wurde bereits behandelt. Zu jeder Normalform gibt es eine Kurzform:

Kurzform	Normalform	Bemerkung
Bitoperationen		
x &= 4	x = x & 4	x wird mit der Zahl 4 über bitorientiert-UND verknüpft und x zugewiesen.
x  = 6	x = x   6	x wird mit der Zahl 6 über ODER verknüpft und x zugewiesen.
x ^= 5	x = x ^ 5	x wird mit der Zahl 5 über EXCLUSIV-ODER verknüpft und x zugewiesen.
x <<= 1	x = x << 1	Linksschieben von x um eine Stelle und x zuweisen
x >>= 1	x = x >> 1	Rechtsschieben von x um eine Stelle und x zuweisen
Arithmetische Operationen		
x += y	x = x + y	x wird um den Wert von y erhöht.
x -= y	x = x - y	x wird um den Wert von y verringert.
x *= y	x = x * y	x wird das Produkt aus x und y zugeordnet.
x /= y	x = x / y	x wird der Quotient aus x und y zugeordnet.
x %= y	x = x % y	x wird der Rest der Ganzzahldivision von x und y zugewiesen.
x ++	x = x + 1	x wird um 1 erhöht (Inkrement-Operator).
x --	x = x - 1	x wird um 1 verringert (Dekrement-Operator).

[LUH17]

# 1. Operatoren

14

## Umwandlungsoperator ()

Werden bei einer Operation (Zuweisungsoperation, Arithmetische Operationen, ...) unterschiedliche Datentypen (Variablen und Konstanten) verwendet, so kann dies zu falschen Ergebnissen führen, da einzelne Datentypen intern umgewandelt werden. Jede Operation kann dabei nur auf einem gleichen Datentyp erfolgen, daher ist eine **Typumwandlung** (engl. **Cast**) notwendig. Grundsätzlich gibt es zwei Möglichkeiten:

### Implizite Casts

Hier überlässt man die Typumwandlung dem Compiler. Dieser hat dafür Regeln hinterlegt, welche meist dem Programmierer nicht immer bekannt sind. **->Hohe Fehleranfälligkeit**

```
iVal = dVal + 1;           // <int> = <double> + <int>
uiVal2 = uiVal3 + iVal;    // <unsigned int> = <unsigned int> + <int>
dVal = 5/2;                // <double> = <int> / <int>
```

### Explizite Casts

Hier erfolgt die Typumwandlung durch den Programmierer. Siehe C-Coding Styleguide!

```
iVal = (int)dVal + 1;       // <int> = <int> + <int>
dVal = (double)5/(double)2; // Alternative: dVal = 5./2.;
```

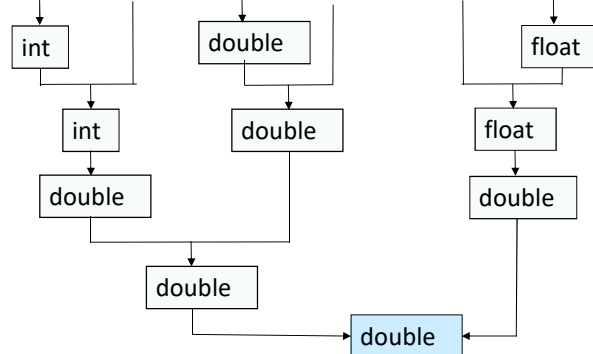
## 1. Operatoren

15

### Typumwandlung – Implizite Casts

```
char cVal = 'C';
int iVal = -73;
float fVal = 4.73f;
double dVal = 4.739991;
double dRes;
```

```
dRes = (cVal / iVal) + (fVal * dVal) - (fVal + iVal);
```



#### Falls

Konstanten und Variablen mit unterschiedlichem Datentyp in einer Operation verwendet werden,

#### dann

werden diese in den höheren Typ umgewandelt (nur innerhalb der Operation).

#### Achtung:

Bei unsigned int und int ist unsigned int der höhere Typ!

Dto. für char/unsigned char, short int/unsigned short int usw.

## 1. Operatoren

16

### Typumwandlung – Explizite Casts – Operator (<Datentyp>)

Falls die Umwandlung eines Datentyps notwendig ist, sollte dies durch einen expliziten Cast geschehen. Dadurch wird angezeigt, dass dies so vom Programmierer gewünscht wurde. Ob dies dann auch immer korrekt ist, ist nicht gegeben.

C-Coding Styleguide: Regel DV13: „Implizite Casts [...] sind zu vermeiden. Nur explizite Casts verwenden [...]“

```
char cVal = 'C';
int iVal = -73;
float fVal = 4.73f;
double dVal = 4.739991;
double dRes;
```

```
dRes = (double)((int)cVal / iVal) +
        ((double)fVal * dVal) -
        (double)(fVal + (float)iVal);
```

Bei literalen Konstanten kann schon im Vorfeld der richtige Typ ausgewählt werden:

```
float fVal = 4.73F;
```

Andere Möglichkeit ist zu umständlich:

```
float fVal = (float)4.73;
```



## 2. Formatierte Ausgabe

17

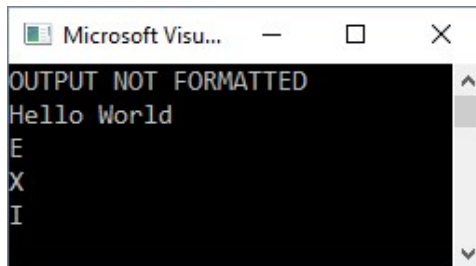
### Unterscheidung

- Ausgabe (nicht formatiert)
- Formatierte Ausgaben

#### Nicht formatierte Ausgabe:

Mittels der Funktion `putchar` kann ein Zeichen auf der Konsole ausgegeben werden. Die Funktion `puts` (s für String, dt. Zeichenkette) wird verwendet, um eine Zeichenkette auf der Konsole auszugeben. Beide Funktionen berücksichtigen keine Formatierung. Die Funktion `puts` fügt automatisch am Ende ein CRLF (Carriage Return (and) Line Feed) ein.

```
unsigned char ucChar = 'E';
puts("OUTPUT NOT FORMATTED");
puts("Hello World");
putchar(ucChar);
putchar('\n');
putchar('X');
putchar('\n');
putchar(73);
putchar('\n');
```



## 2. Formatierte Ausgabe

18

### Allround-Funktion printf

Mit `printf` kann eine formatierte Ausgabe realisiert werden. Es muss eine Zeichenkette als literale Konstante vorgegeben werden, in die durch Formatbezeichner Werte von Variablen eingebettet werden. Die **wichtigsten** Formatbezeichner wurden schon genannt:

%i oder %d	int (vorzeichenbehaftet)
%c	char
%f	float <b>und</b> double (C89)
%e oder %E	float <b>und</b> double in Exponentialformat
%s	Zeichenkette
%x oder %X	Hexadezimalzahl (Ziffernbuchstaben groß (%X) oder klein (%x))

Weitere finden sich in [LUH17].

## 2. Formatierte Ausgabe

19

### Angabe Modus

In einen Formatbezeichner kann noch ein **Modus** eingebettet werden. Hier zwischen % und f.

Zeichen	Beispiel: <code>printf("%s-5.2f");</code>
-	linksbündig
5	5 Stellen insgesamt <b>mindestens!</b>
2	2 Stellen nach dem Komma

Zeichen	Zweck
-	Linksbündig
+	Das positive Vorzeichen wird auch ausgedruckt.
#	Auswahl einer alternativen Darstellung des Ausgabewertes: > Vor Hexadezimalzahlen wird 0x eingefügt. > Gleitkommazahlen erhalten auch dann einen Dezimalpunkt, wenn keine Stellen nach dem Komma folgen.
0	Führende Nullen werden ausgegeben.

[LUH17]

## 2. Formatierte Ausgabe

20

### Steuersequenzen

In die Zeichenkette (literale Konstante) von printf können noch Steuersequenzen („nicht druckbare Zeichen“) eingebettet werden.

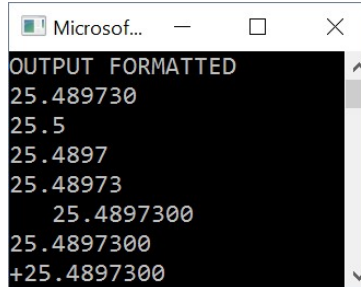
Code	Bedeutung
\a	BEL (bell) – akustisches Warnsignal
\b	BS (backspace) – setzt Cursor um eine Position nach links
\f	FF (formfeed) – ein Seitenvorschub wird ausgelöst.
\n	NL (newline) – Cursor geht zum Anfang der nächsten Zeile
\r	CR (carriage return) – Cursor springt zum Anfang der aktuellen Zeile
\t	HT (horizontal tab) – Zeilenvorschub zur nächsten horizontalen Tabpos.
\v	VT (vertical tab) – Cursor springt zur nächsten vertikalen Tabulatorposition
\"	" wird ausgegeben
\'	' wird ausgegeben
\?	? wird ausgegeben
\\	\ wird ausgegeben
\0	Endmarkierung einer Zeichenkette
\ooo	Zeichen, das der Oktalzahl ooo entspricht (\123 → P)
\xhh	Zeichen, das der Hexadezimalzahl hh entspricht (\x50 → P)

## 2. Formatierte Ausgabe

21

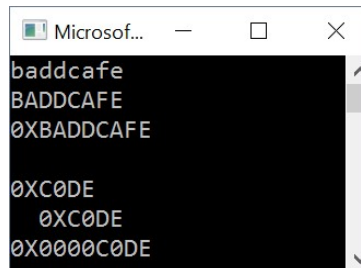
### Formatierte Ausgabe - Codebeispiele

```
dVal = 25.48973;
printf("OUTPUT FORMATTED\n");
printf("%f\n", dVal);
printf("%.1f\n", dVal);
printf("%.2f\n", dVal);
printf("%.3f\n", dVal);
printf("%.13f\n", dVal);
printf("%.13f\n", dVal);
printf("%.13f\n", dVal);
```



```
Microsoft...
OUTPUT FORMATTED
25.489730
25.5
25.4897
25.48973
25.4897300
25.4897300
+25.4897300
```

```
uiVal = 0xBADDCAFE;
printf("%x\n", uiVal);
printf("%X\n", uiVal);
printf("%#X\n", uiVal);
uiVal = 0xC0DE;
printf("\n%#X\n", uiVal);
printf("%#8X\n", uiVal);
printf("%#010X\n", uiVal);
```



```
Microsoft...
baddcafe
BADDCAFE
0XBADDCAFE
0XC0DE
0XC0DE
0X0000C0DE
```

## 3. Formatierte Eingabe

22

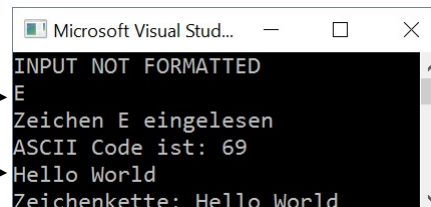
### Unterscheidung

- Eingabe (nicht formatiert)
- Formatierte Eingabe

#### Nicht formatierte Eingabe:

Mittels der Funktion `getchar` kann ein Zeichen von der Konsole eingelesen werden. Die Funktion `gets` (s für String, dt. Zeichenkette) wird verwendet, um eine Zeichenkette von der Konsole einzulesen. `gets` wurde mit C11 aus dem Standard entfernt. Beide Funktionen berücksichtigen keine Formatierung. Details zum Umgang mit Zeichenketten werden später behandelt. Eingabe mit Returntaste abschließen!

```
//Input - not formatted
puts("INPUT NOT FORMATTED");
iCh = getchar();
printf("Zeichen %c eingelesen\n", iCh);
printf("ASCII Code ist: %d\n", iCh);
getchar(); // Clear Keyboard Buffer
gets(acStr);
printf("Zeichenkette: %s", acStr);
```



```
Microsoft Visual Stud...
INPUT NOT FORMATTED
E
Zeichen E eingelesen
ASCII Code ist: 69
Hello World
Zeichenkette: Hello World
```

### 3. Formatierte Ausgabe

23

#### Allround-Funktion scanf

Mit scanf kann eine formatierte Eingabe realisiert werden. Es kann eine Zeichenkette als literale Konstante vorgegeben werden, in die durch Formatbeschreiber Werte von Variablen eingebettet werden. Die **wichtigsten** Formatbezeichner wurden schon genannt:

%i oder %d	int (vorzeichenbehaftet)	
%c	char	
%f	float	<b>Hinweis: Unterscheidung für double und float!</b> Bei printf war Formatbezeichner für double und float gleich (%f für float und double). Bei scanf ist dies unterschiedlich.
%lf	double	
%s	Zeichenkette	

Wie auch schon bei gets und getchar ist die Eingabe mit der Returnntaste abzuschließen.

### 3. Formatierte Ausgabe

24

#### Herausforderungen scanf

Die Funktion scanf ist sehr fehleranfällig und gilt auch als „unsicher“ (siehe später).

```
int iVal;  
scanf("%i", &iVal);
```

Vor Variablennamen muss ein **&** vorangestellt werden (siehe später)

Hier ein paar Tipps zur Fehlervermeidung und Handhabung:

- Immer nur eine Variable pro scanf einlesen
- Keine weiteren Zeichen in die literale Zeichenkonstante einfügen (nur ein Formatbezeichner)
- Ein vorheriges printf oder puts kann dem Anwender anzeigen, was einzugeben ist.
- Der Formatbezeichner %c hinterlässt im Eingabebuffer noch ein CRLF. Der Eingabebuffer kann durch ein zusätzliches getchar geleert werden.
- Nie das &-Zeichen vergessen.
- SDL Checks sind auszuschalten (Project -> Properties -> C/C++ -> General -> SDL checks -> no(/sdl-), ansonsten wird das Projekt wegen scanf nicht übersetzt.

### 3. Formatierte Eingabe

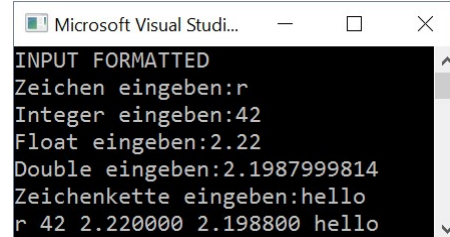
25

#### Formatierte Eingabe - Codebeispiele

```
//Input - formatted
puts("\nINPUT FORMATTED");
printf("Zeichen eingeben:");
scanf("%c", &cVal);
getchar();
printf("Integer eingeben:");
scanf("%i", &iVal);
printf("Float eingeben:");
scanf("%f", &fVal);
printf("Double eingeben:");
scanf("%lf", &dVal);
printf("Zeichenkette eingeben:");
scanf("%s", &acStr);
printf("%c %i %f %f %s",
       cVal, iVal, fVal, dVal, acStr);
```

Hinweis: Bei %s kann bei scanf auf das & verzichtet werden.

```
scanf("%s", acStr);
```



Fließkommazahlen (float und double) werden intern in einen genormten Format dargestellt (IEEE754). Nicht jede reelle Zahl kann dargestellt werden (Rundungsungenauigkeiten entstehen). Siehe E-Learning Modul -> Rat. Zahlen -> Gleitpunktzahlen (2)

### Zusammenfassung KE 2

26

#### Behandelte Schlüsselwörter in KE 2

Schlüsselwörter C89:

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

Schlüsselwörter ab C99:

_Bool✓	_Complex✓	_Imaginary✓	inline	restrict
--------	-----------	-------------	--------	----------

Schlüsselwörter ab C11:

_Alignas	_Alignof	_Atomic	_Generic	_Noreturn
_Static_assert	_Thread_local			

Keine neuen Schlüsselwörter in KE 2 behandelt.