Kapitel 5

Datentypen und Operatoren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 5 – Datentypen und Operatoren

5.1. Elementare Datentypen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Elementare Datentypen (int, float usw.) sind bereits in der Sprache vorgegeben und es gibt Operationen für diese Datentypen.

Zusammengesetzte Datentypen werden vom Programmierer definiert, zum Beispiel:

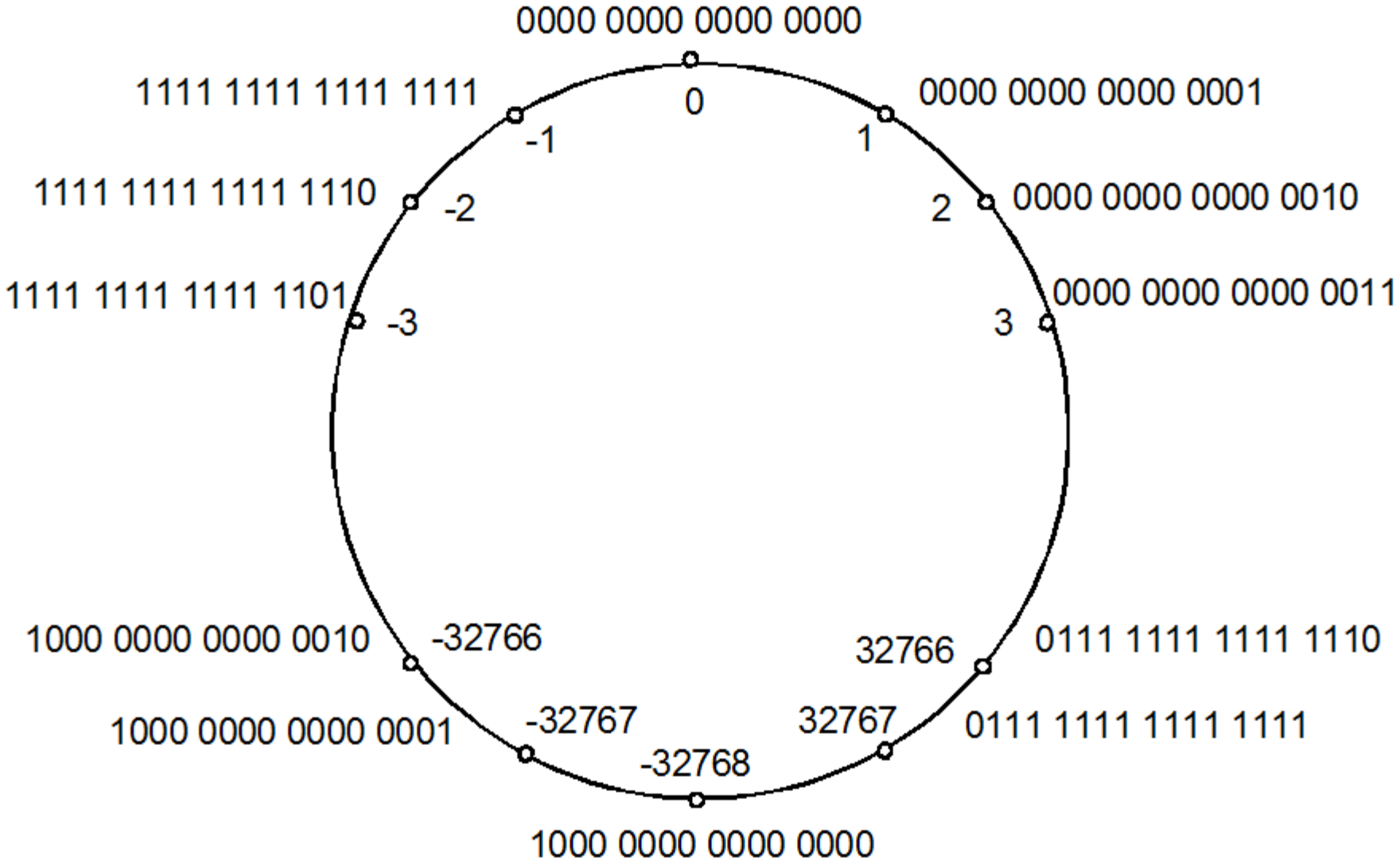
- um komplexe Zahlen darzustellen
  (Kombination zweier float-Zahlen → Addition ist komplizierter),
- um einen Studenten zu beschreiben
  (setzt sich ebenfalls aus elementaren Datentypen zusammen).

Eigenschaften eines Datentyps:

- Wertebereich, Nachkommastellen (bei Gleitkommazahlen)
- Speicherplatzbedarf
- Interne Darstellung / Speicherung
- Gültige Operationen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Ganze Zahlen mit Vorzeichen (a):



Ganze Zahlen mit Vorzeichen (b):

Neben dem Datentyp int gibt es weitere Typen zur Darstellung vorzeichenbehafteter ganze Zahlen. Diese decken unterschiedliche Wertebereiche ab (Kompromiss: Speicherplatz/Wertebereich/ Rechenzeit):

short	`short i;`
int	`int i;`
long	`long i;`
long long	`long long i;`

Zahlenbereich bei n Bit: $(-2^{n-1})\ldots(+2^{n-1}-1)$

n = 16, 32, 64 - 2, 4, 8 Bytes Speicherplatz
positive Zahlen : höchstwertiges Bit 0
negative Zahlen : höchstwertiges Bit 1

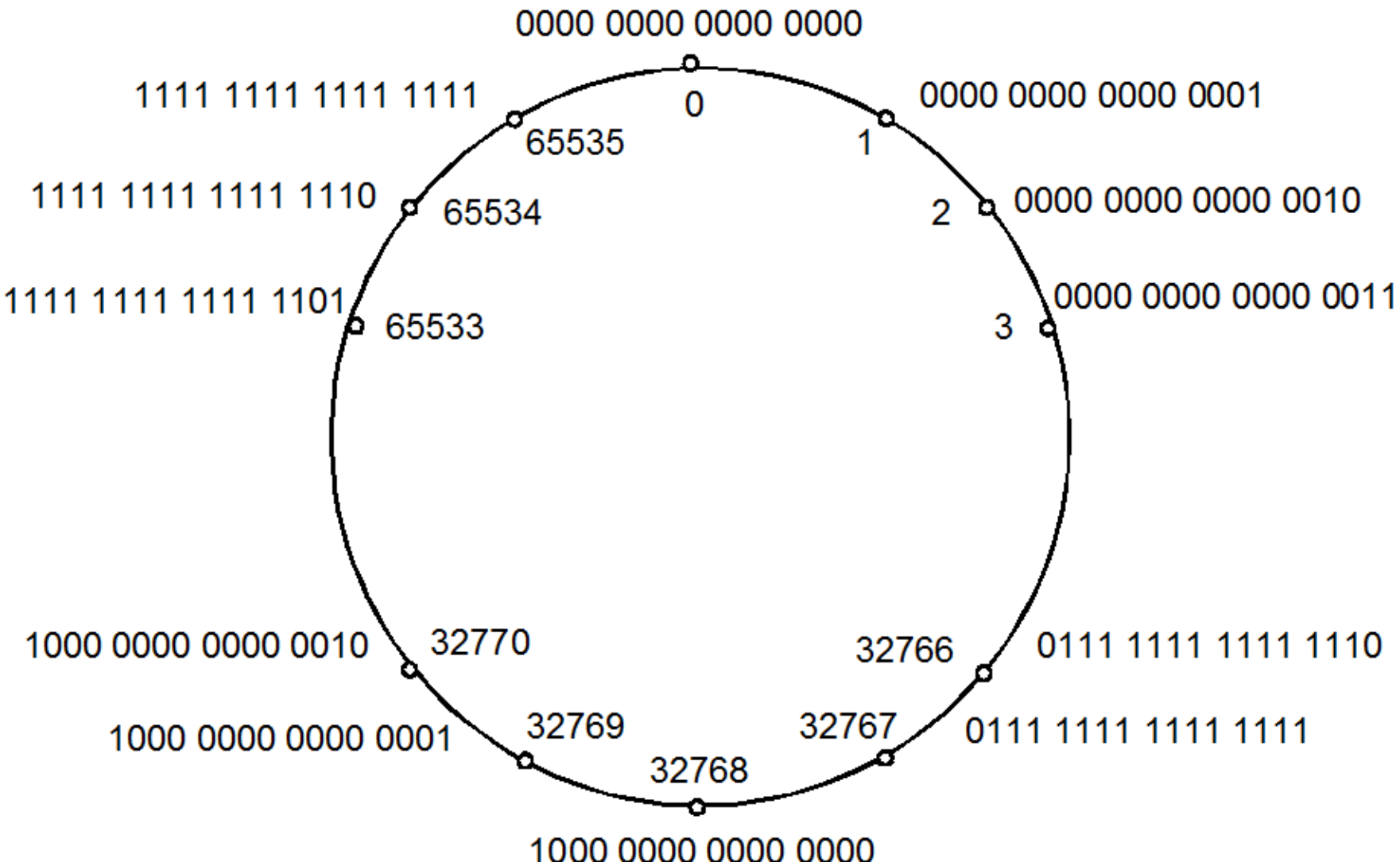
Ganze Zahlen mit Vorzeichen (c):

Größe in Bytes	Linux/GCC	LCC-WIN32	Visual C++
short	2	2	2
int	4	4	4
long	4	4	4
long long	8	8	8

Größe in Bytes	Wertebereich
2 Byte / 16 Bit	-32.768 … +32.767
4 Byte / 32 Bit	-2.147.483.648 … +2.147.483.647
8 Byte / 64 Bit	$-9{,}2 \times 10^{18}$ … $+9{,}2 \times 10^{18}$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Vorzeichenlose ganze Zahlen (a):



Vorzeichenlose ganze Zahlen (b):

unsigned short	`unsigned short i;`
unsigned int	`unsigned int i;`
unsigned long	`unsigned long i;`
unsigned long long	`unsigned long long i;`

```
#include <stdio.h>
int main(void)
{
        unsigned short      s = -1;
        unsigned int        i = -1;
        unsigned long long  l = -1;

        printf("%d, %x  \n", sizeof(s), s);
        printf("%d, %x  \n", sizeof(i), i);
        printf("%d, %llx\n", sizeof(l), l);
        return 0;
}
```

Wie lautet die Ausgabe dieses Programms…?

Vorzeichenlose ganze Zahlen (c):

Größe in Bytes	Linux/GCC	LCC-WIN32	Visual C++
unsigned short	2	2	2
unsigned int	4	4	4
unsigned long	4	4	4
unsigned long long	8	8	8

Größe in Bytes	Wertebereich
2 Byte / 16 Bit	0 … 65.535
4 Byte / 32 Bit	0 … 4.294.967.295
8 Byte / 64 Bit	0 … $18{,}4 \times 10^{18}$

Fließkommazahlen, Gleitkommazahlen (a):

Fließkommazahlen werden benötigt für:

- Sehr große und sehr kleine Zahlen
- Zahlen mit Nachkommastellen

Fließkommazahlen werden **völlig anders behandelt** als ganze Zahlen. Für jede Fließkommazahl werden 32 oder 64 Bit Speicherplatz benötigt. Eine Fließkommazahl wird dabei als Kombination von **Vorzeichen, Exponent und Mantisse** gespeichert.

Typen:

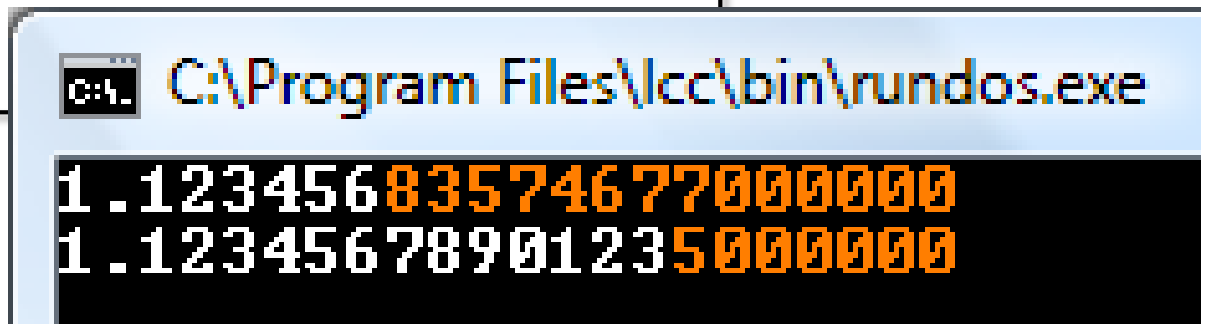
`float`	einfache Genauigkeit	(4 Byte / 32 Bit)
Zahlenbereich (jeweils pos./neg.)		$1{,}2 \times 10^{-38}$ … $3{,}4 \times 10^{+38}$
`double`	doppelte Genauigkeit	(8 Byte / 64 Bit)
Zahlenbereich (jeweils pos./neg.)		$2{,}2 \times 10^{-308}$ … $1{,}8 \times 10^{+308}$

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Fließkommazahlen, Gleitkommazahlen (b):

Bei der Arbeit mit Fließkommazahlen ist deren **begrenzte Genauigkeit** zu beachten! Beim Datentyp float werden 6-7 **signifikante Stellen** berücksichtigt, bei double sind es 15-16 Stellen.

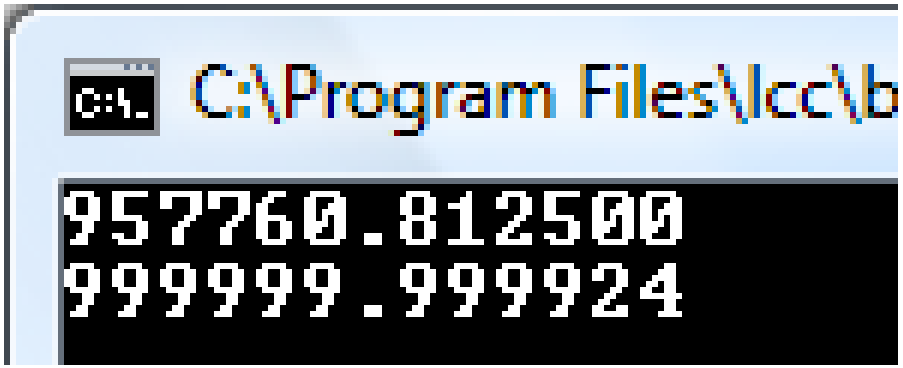
```
#include <stdio.h>
int main(void)
{
        float  f = 1.12345678901234567890;
        double d = 1.12345678901234567890;
        printf("%.20f\n%.20f\n", f, d);
        return 0;
}
```



Fließkommazahlen, Gleitkommazahlen (c):

```c
/* 7.000.000 x 1/7 berechnen */
#include <stdio.h>
int main(void)
{
  float  f = 1.0 / 7.0, fsum = 0;
  double d = 1.0 / 7.0, dsum = 0;
  long l;
  for(l = 1; l <= 7000000; l = l + 1)
  {
    fsum = fsum + f;
    dsum = dsum + d;
  }
  printf("%f\n", fsum);
  printf("%f\n", dsum);
  return 0;
}
```

Achtung: Rundungsfehler können sich aufsummieren!



Buchstaben, Ziffern, Sonderzeichen… (a):

Um einzelne Zeichen zu speichern gibt es einen eigenen Datentyp **char** mit einer Größe von 8 Bit (1 Byte). Zur Bearbeitung von Buchstaben, Ziffern und Sonderzeichen wird dabei in der Regel ein erweiterter ASCII-Code benutzt (z. B. ISO-Latin1).

```
#include <stdio.h>
int main(void)
{
   char c;
   c = 'C';
   c = c - 2;
   printf("%c hat ASCII-Code %d\n", c, c);
   return 0;
}
```

Definition einer char-Variablen

Zeichenkonstanten werden mit einfachen Anführungszeichen geschrieben

Mit Zeichen (bzw. deren ASCII-Code) kann man auch rechnen

Das Formatelement („Platzhalter“) zur Ausgabe eines Zeichens ist %c

Buchstaben, Ziffern, Sonderzeichen… (b):

```
char y = 5;  /* ASCII-Code Steuerzeichen "ENQ" */
printf("y = %d", y);
printf("y = %c", y);

y = '5';  /* y bekommt den Wert 53 */
printf("y = %d", y);
printf("y = %c", y);
```

Ausgabe: `y = 5`
`y =`

Ausgabe: `y = 53`
`y = 5`

Beispiele:

1) Prüfen, ob char-Variable gleich 'j' ist

2) Prüfen, ob char-Variable gleich 'j' oder 'J' ist

3) Prüfen, ob char-Variable Großbuchstaben enthält

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 5 – Datentypen und Operatoren

Symbolische Konstanten sind Konstanten, die einen Namen besitzen. Meist werden sie am Programmanfang definiert; dies ist allerdings nicht zwingend erforderlich.

- Die Anweisung **#define** definiert eine symbolische Konstante (gefolgt von Name und Wert der Konstanten)
- Symbolische Konstanten müssen auf jeden Fall vor der ersten Verwendung definiert werden.
- Konvention: **Namen in Großbuchstaben** schreiben

Beispiele:

```
#define PI 3.14159265
#define BAUTEIL_EINZELPREIS 10.0
#define BEARBEITER "Tom"
```

5.2. Symbolische Konstanten

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

```
#include <stdio.h>
#define PI 3.14159265
int main(void)
{
  int wahl; float r, erg;
  printf("Umfang(1)-Flaeche(2)-Volumen(3)?"); scanf("%d", &wahl);
  printf("Radius ?\n"); scanf("%f", &r);
  switch(wahl)
  {
  case 1:  erg = 2.0*PI*r;
           printf("Umfang = %f", erg);
           break;
  case 2:  erg = PI*r*r;
           printf("Flaeche = %f", erg);
           break;
  case 3:  erg = 4.0/3.0*PI*r*r*r;
           printf("Volumen = %f", erg);
           break;
  default: printf("Auswahl falsch");
  }
  return 0;
}
```

Symbolische Konstanten erhöhen die Lesbarkeit eines Programms. Sie erleichtern außerdem nachträgliche Änderungen des Quelltextes.

Symbolische Konstanten werden häufig in **Include-Dateien** (Header-Dateien) aufgelistet. Solche Dateien können leicht von anderen Programmen (weiter-)verwendet werden. Viele Include-Dateien werden schon vom Compiler-Hersteller mitgeliefert.

Beispiel: Ausschnitt aus der Include-Datei math.h

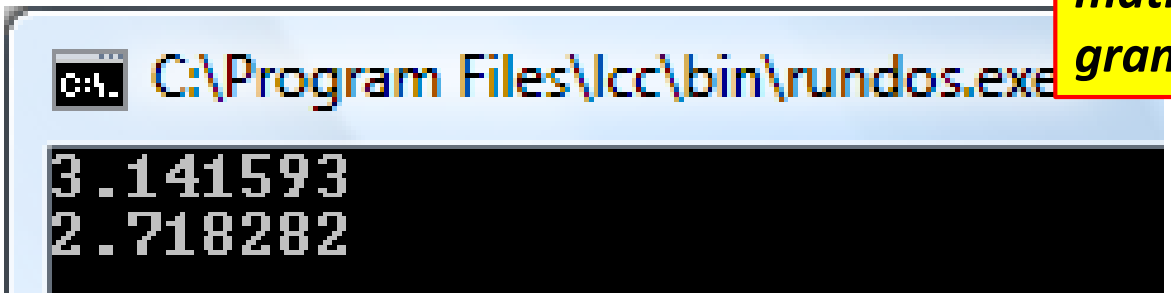
```
…
/* Useful constants. */
#define M_E         2.7182818284590452354
#define M_LN2       0.69314718055994530942
#define M_LN10      2.30258509299404568402
#define M_PI        3.14159265358979323846
#define M_PI_2      1.57079632679489661923
…
```

```
#define _USE_MATH_DEFINES
#include <math.h>
#include <stdio.h>
int main(void)
{
    printf("%f \n", M_PI);
    printf("%f \n", M_E );
    return 0;
}
```

Fehlt diese Definition, werden bei manchen Compilern die Konstanten M_PI, M_E (usw.) nicht erkannt!

Der Inhalt der Datei math.h wird gelesen, wenn das C-Programm compiliert wird.

Alle Konstanten in der Datei math.h können jetzt im Programm verwendet werden.



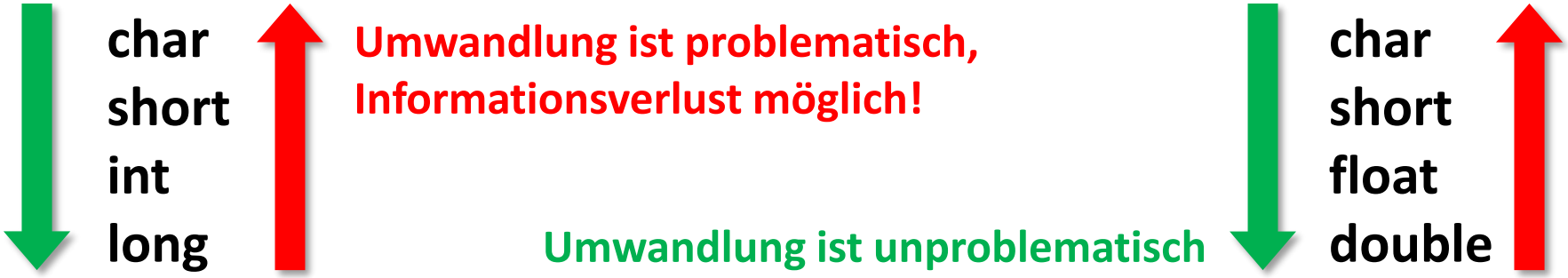
Kapitel 5 – Datentypen und Operatoren

5.3. Typumwandlungen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Im Rechner werden ganze Zahlen und Fließkommazahlen völlig unterschiedlich verarbeitet.

- Darf man einfach einer Fließkommavariablen den Wert einer Integer-Variablen zuweisen?
- Ist das Umgekehrte ebenfalls möglich?
- Was passiert mit den Nachkommastellen einer Fließkommazahl, wenn man sie einer Integer-Zahl zuweist?
- Welche anderen Probleme können auftreten?



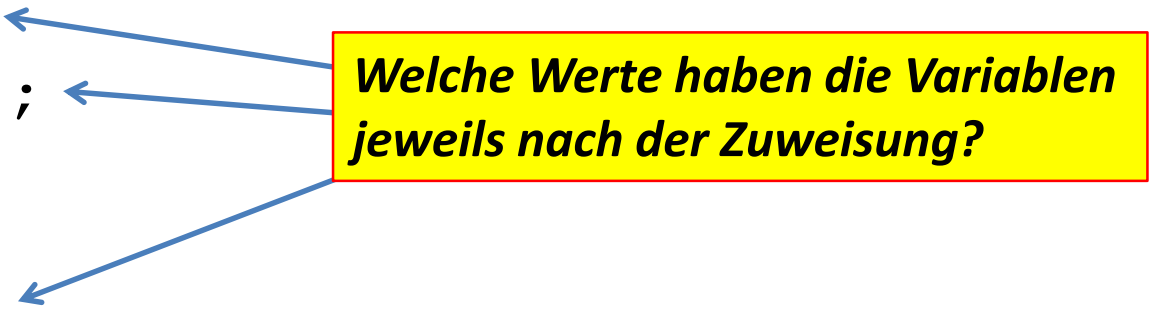
HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Implizite Typumwandlung:

Bei der impliziten Typumwandlung wird die Umwandlung vom Programmierer nicht „ausdrücklich verlangt“. Sie wird vom Compiler automatisch anhand der Datentypen von Variablen und Ausdrücken erkannt und „implizit“ durchgeführt

```
int i1, i2;
double d1, d2, d3, d4;

i1 = 3;
d1 = i1 / 4;
d2 = i1 / 4.0;
d3 = 7.6;
d4 = 3.5;
i2 = d3 + d4;
```



Explizite Typumwandlung:

Anders als bei der impliziten Typumwandlung wird die explizite Typumwandlung im Quelltext angegeben. Es gilt folgende Syntax:

```
(gewünschter Typ)Ausdruck
```

```
int i1, i2, i3;
double d1, d2, d3, d4;

i1 = 3;
d1 = (double)i1 / 4;
d2 = i1 / (double)4;
d3 = (double)(i1 / 4);
d4 = 1.75;
i2 = (int)(d4 + d4);
i3 = (int)d4 + (int)d4;
```

Welche Werte haben die Variablen jeweils nach der Zuweisung?

Kapitel 5 – Datentypen und Operatoren

Operatoren nach Funktionen geordnet:

Funktion	Operatoren
arithmetisch	+ - * / % ++ --
relational	> >= < <= == !=
logisch	&& || !
bitorientiert	& | ^ ~ << >>
zeigerorientiert	& * ->
zuweisend	= += -= *= /= %= &= ^= |= <<= >>=
sonstige	, (type) ( ) [ ] sizeof ?: .

Arithmetische Operatoren:

`+ - * /`	Addition, Subtraktion, Multiplikation, Division
`%`	Modulo-Operator (Rest einer Integer-Division)
`++ --`	Inkrement-, Dekrement-Operatoren (Wert einer Variablen um 1 erhöhen, erniedrigen)

Inkrement-Operatoren können als Präfix- (++a) oder als Postfix-Operatoren (a++) angewendet werden. Welcher Unterschied besteht zwischen beiden Varianten?

```
int  x, y, zahl;
zahl = 5;
++zahl;
zahl++;
x = ++zahl;
y = zahl++;
```

Welche Werte haben die Variablen jeweils nach der Ausführung dieser Programmzeilen?

Relationale Operatoren:

> >= <= <	größer, größer/gleich, kleiner/gleich, kleiner
==	Vergleich auf Gleichheit
!=	Vergleich auf Ungleichheit

Logische Operatoren:

&& ||	Und- bzw. Oder-Verknüpfung
!	Negation

```
if( (x >= 10) && (x <= 20) )
        printf("x im Bereich 10...20");

if( !(x == y) )
        printf("x nicht gleich y");
```

Zuweisende Operatoren:

=	Zuweisungsoperator (keine math. Gleichheit)
+= -=	Kurzschreibweisen für Addition, Subtraktion
*= /=	Kurzschreibweisen für Multiplikation, Division
%=	Kurzschreibweise für Modulo-Operator

```
zahl += 10          zahl = zahl + 10
zahl %= 10          zahl = zahl % 10
```

Inkrementierung einer Integer-Zahl:

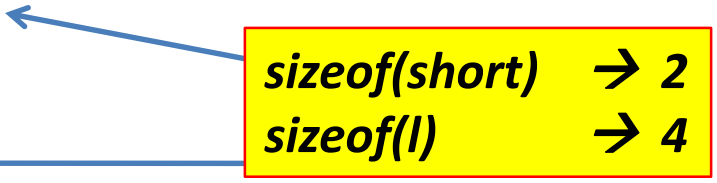
```
zahl = zahl + 1     zahl += 1
zahl++              ++zahl
```

Sonstige Operatoren:

,	Aufzählungsoperator zur Verkettung von Variablen oder Ausdrücken, z. B. in einer for-Anweisung
sizeof()	Dieser „Operator“ liefert die Größe (in Bytes) einer Variablen oder eines Datentyps.

```
int a, b;
for(k=0, j=2; k<=100; k++, i+=7)
          ...

i = sizeof(short);
long l;
i = sizeof(l);
```



Operator
( ) [ ] -> .
! ~ ++ -- + - * & (type) sizeof
* / %
+ -
<< >>
< <= > >=
== !=
&
^
|
&&
||
? :
= += -= *= /= %= &= ^= |= <<= >>=
,

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Operatoren nach Priorität geordnet