3. Datentypen

3.1. Information

Information ist eine physikalische Größe und gibt das Wissen über einen gewissen Zustand an.
Die Einheit der Information ist das Bit.
Ein Bit kann genau zwei Zustände annehmen, nämlich den Zustand 0 oder 1, oder anders ausgedrückt ein Zustand kann entweder wahr oder falsch sein.

Will man mehr als zwei Zustände darstellen muss man mehrere Bits benutzen.
Die allgemeine Formel um aus der Bitanzahl die Anzahl der Zustände zu gewinnen lautet:

$2^{\text{Anzahl der Bits}}$ = Anzahl der Zustände

Anzahl der Bits = $\log_2$ Anzahl der Zustände

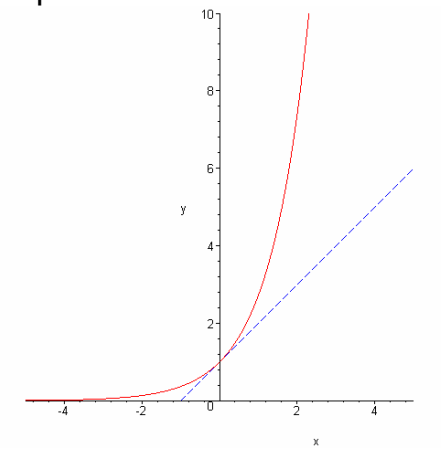
Umrechnung zwischen Logarithmensystemen

Da mit dem Taschenrechner nur dekadische und natürliche Logarithmen berechenbar sind, ist es von Fall zu Fall Logarithmen umzurechnen.

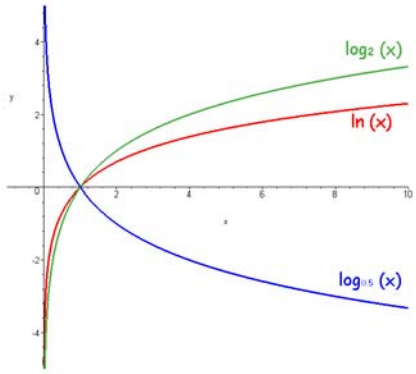
$$\log_a b = \frac{\lg b}{\lg a} = \frac{\ln b}{\ln a}$$

dabei ist $\log_e (c) = \ln (c)$ der natürlicher Logarithmus zur Basis e
e ist die Eulersche Zahl

Expontentialfunktion



Umkehrfunktion der Exponentialfunktion ist die Logarithmusfunktion



Vielfache eines Bits:

4 Bit = 1 Nibble 16 Zustände
8 Bit = 1 Byte 256 Zustände
10 Bit 1024 Zustände ist ein KBit (Kilobit)
16 Bit = 1 Word 65 536 Zustände
20 Bit 1 048 576 Zustände ist ein MBit (Megabit)
32 Bit = 1 Double Word 4 294 967 296 Zustände

Aufgepasst ein Kilo bzw. ein Mega entspricht in der Informatik nicht den Umrechnungsfaktoren 1 000 bzw. 1 000 000.

In der Informatik ist die Zahlenwelt immer eine Potenz der Zahl 2.

$2^0 = 1$
$2^1 = 2$
$2^2 = 2 * 2 = 4$
$2^3 = 2 * 2 * 2 = 8 * 1 = 8$
$2^4 = 2 * 2 * 2 * 2 = 8 * 2 = 16$
$2^2 = 2 * 2 * 2 * 2 * 2 = 8 * 4 = 32$
$2^6 = 2 * 2 * 2 * 2 * 2 *2 = 8 * 8 = 64$
$2^7 = 2 * 2 * 2 * 2 * 2 *2 * 2 = 8 * 8 * 2= 128$
$2^8 = 2 * 2 * 2 * 2 * 2 *2 * 2 * 2 = 8 * 8 * 4 = 256$
$2^9 = 2 * 2 * 2 * 2 * 2 *2 * 2 * 2 * 2 = 8 * 8 * 8 = 512$
$2^{10} = 2 * 2 * 2 * 2 * 2 *2 * 2 * 2 * 2 * 2 = 8 * 8 * 8 * 2 = 1024$

3.2. Variablen

Im Programm werden Namen als Variablen eingesetzt, die als Platzhalter für Werte dienen. Variable nehmen im Gegensatz zu Konstanten veränderliche Werte auf. Jede Variable besitzt also einen Namen und einen Wert. Sie stellt somit einen Platzhalter für ihren Wert dar. Der Compiler reserviert entsprechend dem Typ eine bestimmte Menge Speicherplatz für den Platzhalter.

In C unterscheidet man die Definitionen und die Deklarationen von Variablen.

Definitionen

legen die Art der Variablen fest:

den Datentyp,
die Speicherklasse,
Datentyp-Attribute (type qualifier) wie const und volatile fest
sorgen gleichzeitig für die Reservierung des Speicherplatzes.

Deklarationen

legen nur die Art der Variablen bzw. die Schnittstelle der Funktionen, d. h. die Funktionsköpfe, fest. Während Definitionen von Variablen und Funktionen dazu dienen, Datenobjekte bzw. Funktionen im Speicher anzulegen, machen Deklarationen Datenobjekte bzw. Funktionen bekannt, die in anderen Übersetzungseinheiten definiert werden oder in derselben Übersetzungseinheit erst nach ihrer Verwendung definiert werden.

Eine Deklaration umfasst stets den Namen eines Objektes und seinen Typ. Damit weiß der Compiler, mit welchem Typ er einen Namen verbinden muss.

Eine Definition besteht immer aus einer Deklaration und der Reservierung von Speicherplatz.

Wertzuweisung

Die Zuweisung eines Wertes an eine Variable ist eine fundamentale Operation in Computerprogrammen. Eine Variable zeigt eine Verhaltensweise, die einer Wandtafel ähnlich ist: Sie kann jederzeit gelesen werden (sie liefert den Wert) andererseits ausgewischt und überschrieben werden.
Die Zuweisung eines Wertes an eine Variable wird durch das Operatorzeichen“ = „ gemacht.

```
int a;

a = 5;
```

L (left) und R (right) Werte

Ausdrücke habe eine unterschiedliche Bedeutung, je nachdem, ob sie links oder rechts vom Zuweisungsoperator stehen. Wie schon erwähnt steht beim Ausdruck a = b die rechte Seite für einen Wert, während der Ausdruck auf der linken Seite die Stelle angibt, an der der Wert zu speichern ist. Wenn wir das Beispiel noch etwas modifizieren, wird der Unterschied noch deutlicher:

```
a = a + 5 * c
```

Beim Namen a ist rechts vom Zuweisungsoperator der Wert gemeint, der in der Variable mit dem Namen a gespeichert ist, und links ist die Adresse der Variablen a gemeint, in der der Wert des Gesamtausdrucks auf der rechten Seite gespeichert

werden soll. Aus dieser Stellung links oder rechts des Zuweisungsoperators wurden auch die Begriffe L-Wert und R-Wert abgeleitet.

Ein Ausdruck stellt einen L-Wert dar, wenn er sich auf ein Speicherobjekt bezieht. Ein solcher Ausdruck kann links und rechts des Zuweisungsoperators stehen.
Ein Ausdruck, der keinen L-Wert darstellt, stellt einen R-Wert dar. Er darf nur rechts des Zuweisungsoperators stehen. Einem R-Wert kann man also nichts zuweisen.

3.3. Skalare Standarddatentypen

Wie schon erwähnt, ist die summarische Auflistung der Variablen und Konstanten wichtig für die Verständlichkeit des Programms. Insbesondere gehört zu einer guten Dokumentation die explizite Angabe des Wertebereichs. Nachfolgend werden vier äußerst häufige Standardtypen definiert, die in jedem Datenverarbeitungssystem als bekannt vorausgesetzt werden. Trotzdem sind nicht alle Typen auch in jeder Programmiersprache verfügbar. So kennt C nicht den Typ "Boolean". Man kann die fehlenden Typen aber nachbauen.

Boolean

Dieser Typ bezeichnet den Bereich der logischen Werte, bestehend aus zwei Elementen "wahr" ("TRUE", 1) und "falsch" ("FALSE", 0). Auf den Werten dieses Typs sind drei Operatoren definiert: OR (logisches Oder), AND (logisches Und), EXOR (logische Antivalenz), NOT (logisches Nicht). Die Gesetze für diese Operatoren entsprechen jenen der Aussagelogik.

Integer

Dieser Typ bezeichnet den Bereich der ganzen Zahlen. Es sind die folgenden Operatoren definiert:
+ Addition
- Subtraktion
* Multiplikation
/ Ergebnis der ganzzahligenDivision
% Rest der ganzzahligen Division
Bei jedem Computer spezifiziert der Datentyp "Integer" nur eine Untermenge der ganzen Zahlen im mathematischen Sinn, und zwar die Menge aller Zahlen, die betragsmäßig kleiner als ein bestimmter Maximalwert sind. In ISO C werden als Minimalwerte mit zwei Byte definiert:

Typ	Speicherbedarf	Wertebereich
int	2 Byte	-32768 ... 32767
long int	4 Byte	-2147483648 ... 2147483647
short int	2 Byte	-32768 ... 32767
unsigned int	2 Byte	0 ... 65535
unsigned long int	4 Byte	0 ... 4294967295

unsigned short int	2 Byte	0 ... 65535

Negative ganze Zahlen werden nicht durch ein Vorzeichenbit sondern mit dem Zweierkomplement dargestellt.
Die Zahl -4 wird dabei folgendermaßen im Speicher abgelegt.

```
 0100      Stelle die Zahl 4 binär dar
+    1     addiere die Zahl 1
 0101      negiere die Zahl
 1010      diese Zahl entspricht der Zahl -4 im Binärformat
```

Character

Dieser Typ bezeichnet eine endliche, geordnete Menge von Zeichen (Characters). Der Typ Character stützt sich auf den im jeweiligen Computer verwendeten Code zur Darstellung von Zeichen (ASCII, EBCDIC, ANSI). Er ist also nur insofern Standard, als der zugrunde liegende Zeichensatz Standard ist. Mit diesem Typ werden auch zwei vom Zeichensatz abhängige Konversionsmöglichkeiten zwischen "Character" und "Integer" eingeführt (in C "Typecasting"). In ISO C wird als Speichergröße für ein Zeichen mit einen Byte definiert:

Typ	Speicherbedarf	Wertebereich
char	1 Byte	-128 ... 127 oder 0 ... 255
unsigned char	1 Byte	0 ... 255

Um die verschiedenen nationalen Zeichensätze unter einen Hut zu bringen, wurde UNICODE entwickelt. Das hat zur Folge, dass in manchen Programmiersprachen char auch 16 Bit lang sein kann.

Float

Dieser Datentyp stellt nicht Variablen aus dem Wertebereich der reellen Zahlen, sondern Gleitkommazahlen dar. Die Repräsentation dieser Werte hängt auch wieder von der Realisierung im jeweiligen Prozessor ab. Auf dem Datentyp Real sind vier Operatoren definiert:
+ Addition
- Subtraktion
* Multiplikation
/ Division
Anmerkung: Eine explizite Konversion von Integer nach Float existiert nicht. Diese Konversion wird implizit bei der Übersetzung des Ausdrucks vorgenommen. Es gilt für die Wortlänge: float <= double <= long double. In ISO-C wird als Minimum definiert bzw. ist in IEEE 754 für PC Systeme definiert:

Typ	Speicher-bedarf	Wertebereich Vorzeichen beliebig	Exponenten-stellen	Mantissen-stellen

float	4 Byte	$1,5*10^{-45}$ ... $3,4*10^{+38}$	8	23
double	8 Byte	$5,0*10^{-324}$ ... $1,7*10^{+308}$	11	52
long double	10 Byte	$1,9*10^{-4951}$ ... $1,7*10^{+4932}$	15	64

Darstellung einer Gleitkommazahl:

Bei einer Gleitkommazahl wird das Vorzeichen durch ein extra Vorzeichenbit dargestellt. Weiters wird die Zahl in Exponent und Mantisse aufgeteilt.

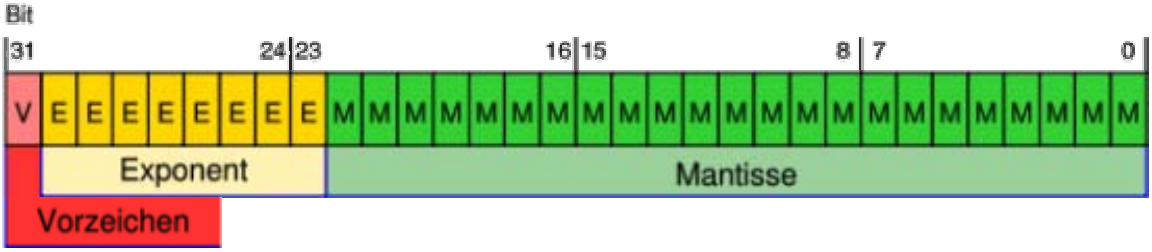


Abbildung 1 Darstellung einer Gleitkommazahl vom Typ float

Die 3,4 wird zuerst in die Zahl 0,34 10^{1} umgewandelt. 34 ist bei dieser Zahl die Mantisse und wird in den Bitbereich der Mantisse gespeichert. 1 ist der Exponent und wird in den Bitbereich des Exponenten gespeichert.

3.4. Formatangaben

Formatangaben werden benötigt um Variablen mit der Funktion printf( )im richtigen Format ausgeben zu können. Sie werden aber auch benötigt um Werte formatiert von der Konsole mit der Funktion scanf( ) lesen zu können.

scanf(“formatstring“,[argument_1, argument_2, ….., argument_n]);

Der formatstring kann alle sichtbaren Zeichen und Formatangaben für die zu lesenden Werte beinhalten. In argument_1 bis argument_n werden die Werte gespeichert. Die Funktion scanf( ) ist genauso wie die Funktion printf( ) in der Funktionsbibliothek „stdio.h“ enthalten.

Formatangaben für printf( )

Formatangabe	Eingabewert
%d	int
%i	int
%u	unsigned int
%o	Oktalzahl
%x	Hexadezimalzahl
%c	char
%f	float, double, long double
%e	float, double, long double in Expontialform
%g	float, double, long double in Exponential- oder Dezimalform
%s	Zeichenkette (String)
%p	Zeiger (Pointer)

Formatangaben für scanf( )

Formatangabe	Eingabewert
%d	int
%hd	short int
%ld	long int
%i	int in Dezimalform, Oktalform mit 0, Hexadezimalform mit 0x
%u	int ohne Vorzeicen
%hu	short int ohne Vorzeichen
%lu	long int ohne Vorzeichen
%o	int in Oktalform
%ho	short in Oktalform
%lo	long in Oktalform
%x	int in Hexadezimalform
%hx	short in Hexadezimalform
%lx	long in Hexadezimalform
%c	char
%f	float
%lf	double
%e	float in Exponentialform
%le	double in Exponentialform
%g	float in Exponential- oder Dezimalform
%Lg	long double
%s	Zeichenkette (String)
%p	Zeiger (Pointer)

Der sizeof – Operator liefert den Speicherbedarf einer Variable oder eines Datentyps zurück.

unsigned int sizeof(argument);

argument kann eine Datentyp oder eine Variable sein.

Bei Gleitkommatypen kann man die Nachkommastellen und die Feldbreite einschränken, dies geschieht indem die Formatangabe folgendermaßen ergänzt:

```
fFloatwert = 3.14159;
printf("Ausgabe: %4.3f",fFloatwert); // Ausgabe: 3.141
```

%4.3 bedeutet, dass die Feldbreite 4 Stellen beträgt und 3 Stellen für Nachkommastellen reserviert sind.

Weiters existieren noch so genannte strukturierte Datentypen wie Felder (Arrays) und Strukturen (Structures) auf diese wird aber am Ende des Jahres eingegangen.

```
#include <stdio.h>

int main(void)
{
```

```
  /* Beachten Sie: Innerhalb eines Paares geschweifter Klammern
*/
  /* immer zuerst alle benoetigten lokalen Variablen
deklarieren */
  /* und dann die Anweisungen schreiben.
*/

  /* Deklaration der (lokalen) Variablen */
  int         i,ii;  /* printf-Format: %d  */
  short int   h,hh;  /* printf-Format: %hd */
  long int    l,ll;  /* printf-Format: %ld */
  char        c,cc;  /* printf-Format: %c  */
  float       f,ff;  /* printf-Format: %f  | %e  | %g  */
  double      d,dd;  /* printf-Format: %lf | %le | %lg */
  long double q,qq;  /* printf-Format: %Lf | %Le | %Lg */

  /* Addition zweier int-Zahlen */
  i=1;
  ii=-2;
  printf("int: i=%d, ii=%d, i+ii=%d\n",i,ii,i+ii);

  /* Ganzzahl-Division zweier short-int-Zahlen */
  h=7;
  hh=2;
  printf("short: h=%hd, hh=%hd, h/hh=%hd\n",h,hh,h/hh);

  /* Divisionsrest zweier long-int-Zahlen */
  l=7l;
  ll=2l;
  /* Achtung beim Prozent: Zur Ausgabe doppelt schreiben */
  printf("long: l=%ld, ll=%ld, l%%ll=%ld\n",l,ll,l%ll);

  /* Ausgabe zweier Zeichen */
  c='A';
  cc='B';
  printf("char: c=%c, cc=%c\n",c,cc);
  /* Probieren Sie auch die Zeichen '\n', '\t', '\b' aus */

  /* Multiplikation zweier float-Zahlen */
  f=1.3E4f;
  ff=-5.7E3f;
  printf("float: f=%f, ff=%f, f*ff=%f\n",f,ff,f*ff);

  /* Division zweier double-Zahlen (Gleitkomma-Division) */
  d=1.3E4;
  dd=-5.7E3;
  printf("double: d=%lf, dd=%lf, d/dd=%lf\n",d,dd,d/dd);

  /* Subtraktion zweier long-double-Zahlen */
  q=1.3E4l;
  qq=1.299999999E4l;
  printf("long double: q=%Lf, qq=%Lf, q-qq=%Lf\n",q,qq,q-qq);

  /* Ergebnis eines Vergleichs ist 0 oder 1 als int-Zahl */
  c='A';
  cc='B';
  printf("Vergleich von %c und %c ergibt %d.\n",c,cc,c==cc);

  /* Mehrere Zuweisungen in einem Ausdruck ausführen */
```

```
  c=cc='X';
  printf("Zwei Ixe: %c%c\n",c,cc);

  /* Man kann Gleitkommazahlen in verschiedenen Formaten
  ausgeben. Die 20 hinter dem % füllt links mit Leerzeichen
  auf 20 Zeichen auf. */
  dd=4E0/3.0;
  printf("Gleit- > %20lf %20le %20lg\n",dd,dd,dd);
  dd=4E9/3.0;
  printf("komma- > %20lf %20le %20lg\n",dd,dd,dd);
  dd=4E20/3.0;
  printf("formate> %20lf %20le %20lg\n",dd,dd,dd);

  /* Funktion main() und damit Programm beenden */
  return(0);
}
```