

Übersicht

Lehrinhalt: Programmierung in C

- Überblick über Programmiersprachen
- **C: Eingebaute Datentypen, Zahlendarstellung, Variablen, Konstanten**
- Operatoren und Ausdrücke
- Anweisungen
- Kontrollstrukturen
- Funktionen
- Zeiger und Felder
- Zeichenketten (Strings)
- Benutzerdefinierte Datentypen
- Dynamischer Speicher
- Dateiarbeit
- Funktionspointer, Rekursion
- Preprozessor

Variablen und Datentypen

Die Verarbeitung erfolgt durch Anweisungen, die die Werte der Variablen verändern.

Ausdrücke enthalten Variablen. Der Wert eines Ausdrucks wird u.a. durch den Wert der enthaltenen Variablen bestimmt.

Variablen können Zahlenwerte, logische Werte (wahr, falsch), Zeichen, Zeichenketten, Zeiger, Felder aufnehmen

Variablen müssen vor ihrer ersten Benutzung auf jeden Fall deklariert worden sein.

Bezeichner (Namen für Variable, Funktionen, ...)

- Namen beginnen mit einem Buchstaben oder einem Unterstrich.
- Die weiteren Zeichen sind Buchstaben, Zahlen oder Unterstriche.
- Schlüsselworte der Sprache dürfen nicht als Name verwendet werden

Variablen und Datentypen

Zulässige Variablennamen:

- a; i; j; k;
- _pointer;
- ganz_langer_name_24_undNochLaenger;
- name; Name; // Groß- und Kleinschreibung wird unterschieden

Nicht als Variablenname zulässig:

- 34Name; // Fehler Zahl am Anfang nicht erlaubt
- Strassen Name; // Leerzeichen nicht erlaubt, besser: Strassen_Name
- Ölinhalt // Fehler, Umlaute verboten
- C&A; // Fehler; Sonderzeichen verboten
- while; // da es ein Schlüsselwort while bereits gibt

Variablenname sollten kurz sein, aber inhaltlich ihre Bedeutung wiedergeben

Variablendeklaration

Alle benutzen Variablen müssen am Anfang des Programms deklariert werden.

Deklaration bedeutet

- Variablennamen erwähnen
- Typ angeben
- Wahlweise eine Initialisierung mit einem Wert

Genereller Aufbau

***typ** variablename;*

***typ** variablename-1, variablename-1;*

***typ** variablename = init-ausdruck;*

Variablendeklaration

Beispiele:

```
int anzahl_patienten;
```

```
float gewicht;
```

```
float x,y,z;
```

```
char eingabe, auswahl='h';
```

```
int anzahl=25;
```

```
int start = anzahl-1;
```

Basisdatentypen

Grunddatentypen

char für einzelne Zeichen im ASCII-Code

int für ganzzahlige Zahlen

float für Gleitkommazahlen (Darstellung reeller Zahlen),

double für Gleitkommazahlen doppelter Genauigkeit

enum für programmspezifische Aufzählungen

Die einige Grunddatentypen können noch mit Modifikatoren versehen werden: *signed, unsigned, short, long*

Basisdatentypen

Typ	Länge	Wertebereich	Genauigkeit
char = signed char unsigned char	1 Byte 1 Byte	$-2^7 \dots +2^7-1$ $0 \dots 2^8-1$	2 Dezimalstellen, z.B. ‚88‘ ist genau
int = signed int unsigned int	4 Byte 4 Byte	$-2^{31} \dots +2^{31}-1$ $0 \dots 2^{32}-1$	9 Dezimalstellen
short int = signed short int unsigned short int	2 Byte 2 Byte	$-2^{15} \dots +2^{15}-1$ $0 \dots 2^{16}-1$	4 Dezimalstellen
long int = signed long int unsigned long int	8 Byte 8 Byte	$-2^{63} \dots +2^{63}-1$ $0 \dots 2^{64}-1$	19 Dezimalstellen (ab C99, 64 Bit)
float double long double	4 Byte 8 Byte 10 Byte	ca. $-10^{38} \dots +10^{38}$ ca. $-10^{308} \dots +10^{308}$ ca. $-10^{4932} \dots +10^{4932}$	6 Dez.-Stellen 15 Dez.-Stellen 19 Dez.-Stellen

Die angegebenen Werte stellen Beispiele für 32-Bit Umgebungen dar, sie sind implementierungsabhängig

Basisdatentypen (Fortsetzung)

		Wertebereich	Genauigkeit
enum { <i>list</i> }	4 Byte	2 ³² verschiedene Werte	
enum <i>id</i> { <i>list</i> }	4 Byte		
enum <i>id</i>	4 Byte		
bool (nur C++)	1 Byte	true, false	
<i>type</i> *	4 Byte	0...2 ³² -1	einzelne Bytes im Speicher adressierbar
void *	4 Byte	0...2 ³² -1	
<i>type</i> & (nur C++)	4 Byte	0...2 ³² -1	

Verwendung der Datentypen

Datentypen für ganze Zahlen in C:

*int, signed int, unsigned int,
long, signed long, unsigned long,
short, signed short, unsigned short*

Beispiel:

*int i = -64;
long li = 3;*

Integer-Variablen werden für beispielsweise für zählbare Dinge benutzt, oder für Index-Berechnungen.

Verwendung der Datentypen

Gleitkommazahlen:

float, double, long double

Beispiele:

double d = 64.3345;

double d1 = 1.234e-22; // ohne Leerzeichen zu schreiben

float f = 67.31f;

float f1 = 2.9744e-22f;

double x = 5.; // 5. entspricht 5.0

Gleitkommazahlen werden für Eigenschaften, Größen verwendet, die als rationale oder reelle Zahlen angegeben werden, z.B. für eine Wachstumsrate von 2.12 %.

Verwendung der Datentypen

Einzelne Zeichen (char) - die Werte entsprechen in der Regel dem ASCII-Zeichensatz

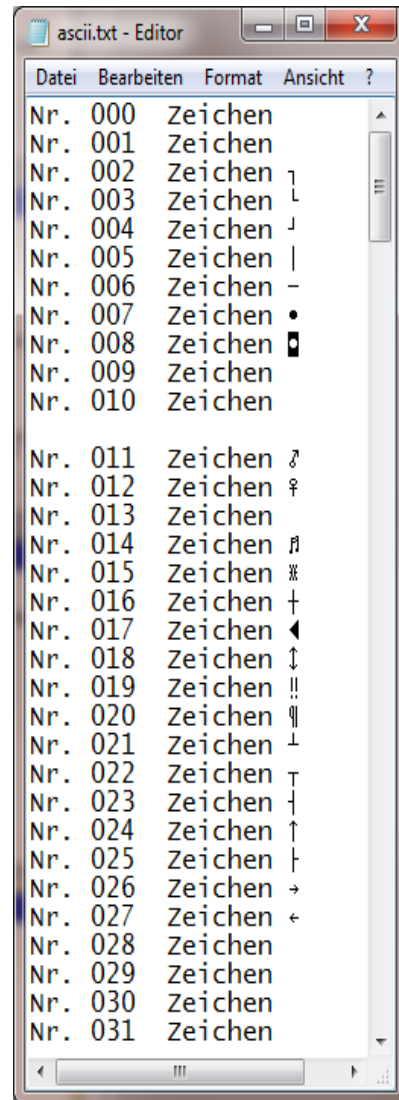
Beispiele:

char c = 64;

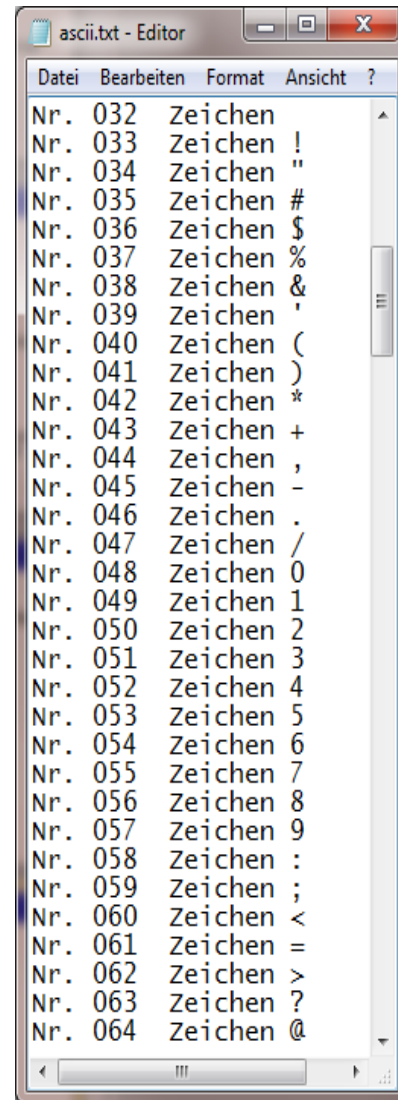
char c1 = 'h';

char c2 = '\n';

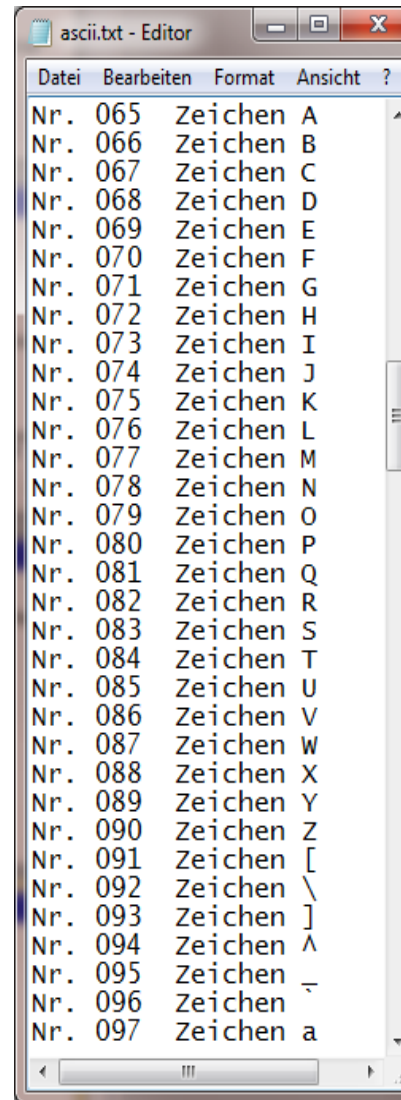
ASCII-Code



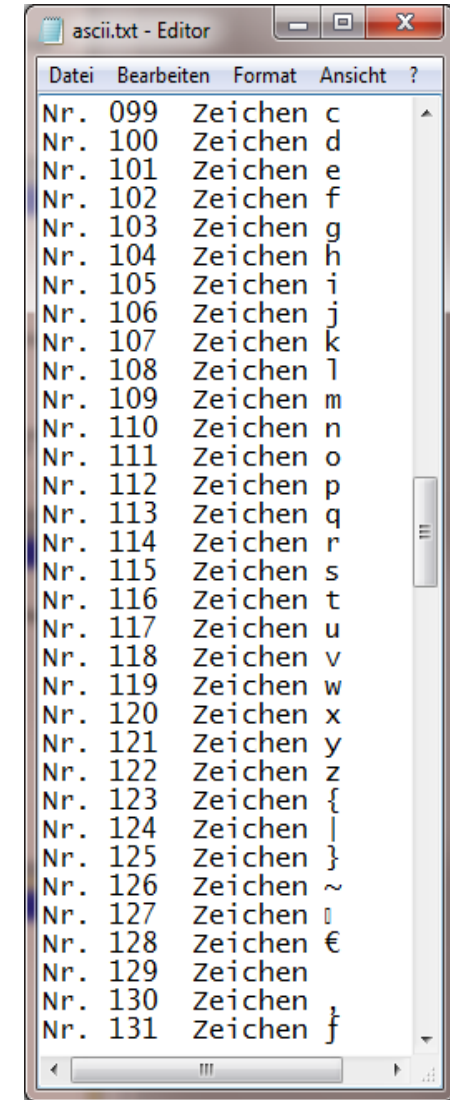
	Datei	Bearbeiten	Format	Ansicht	?
Nr. 000	Zeichen				
Nr. 001	Zeichen				
Nr. 002	Zeichen	␣			
Nr. 003	Zeichen	␣			
Nr. 004	Zeichen	␣			
Nr. 005	Zeichen				
Nr. 006	Zeichen	-			
Nr. 007	Zeichen	•			
Nr. 008	Zeichen	␣			
Nr. 009	Zeichen				
Nr. 010	Zeichen				
Nr. 011	Zeichen	␣			
Nr. 012	Zeichen	␣			
Nr. 013	Zeichen	␣			
Nr. 014	Zeichen	␣			
Nr. 015	Zeichen	␣			
Nr. 016	Zeichen	␣			
Nr. 017	Zeichen	␣			
Nr. 018	Zeichen	␣			
Nr. 019	Zeichen	␣			
Nr. 020	Zeichen	␣			
Nr. 021	Zeichen	␣			
Nr. 022	Zeichen	␣			
Nr. 023	Zeichen	␣			
Nr. 024	Zeichen	␣			
Nr. 025	Zeichen	␣			
Nr. 026	Zeichen	␣			
Nr. 027	Zeichen	␣			
Nr. 028	Zeichen				
Nr. 029	Zeichen				
Nr. 030	Zeichen				
Nr. 031	Zeichen				



	Datei	Bearbeiten	Format	Ansicht	?
Nr. 032	Zeichen				
Nr. 033	Zeichen	!			
Nr. 034	Zeichen	"			
Nr. 035	Zeichen	#			
Nr. 036	Zeichen	\$			
Nr. 037	Zeichen	%			
Nr. 038	Zeichen	&			
Nr. 039	Zeichen	'			
Nr. 040	Zeichen	(
Nr. 041	Zeichen)			
Nr. 042	Zeichen	*			
Nr. 043	Zeichen	+			
Nr. 044	Zeichen	,			
Nr. 045	Zeichen	-			
Nr. 046	Zeichen	.			
Nr. 047	Zeichen	/			
Nr. 048	Zeichen	0			
Nr. 049	Zeichen	1			
Nr. 050	Zeichen	2			
Nr. 051	Zeichen	3			
Nr. 052	Zeichen	4			
Nr. 053	Zeichen	5			
Nr. 054	Zeichen	6			
Nr. 055	Zeichen	7			
Nr. 056	Zeichen	8			
Nr. 057	Zeichen	9			
Nr. 058	Zeichen	:			
Nr. 059	Zeichen	;			
Nr. 060	Zeichen	<			
Nr. 061	Zeichen	=			
Nr. 062	Zeichen	>			
Nr. 063	Zeichen	?			
Nr. 064	Zeichen	@			



	Datei	Bearbeiten	Format	Ansicht	?
Nr. 065	Zeichen	A			
Nr. 066	Zeichen	B			
Nr. 067	Zeichen	C			
Nr. 068	Zeichen	D			
Nr. 069	Zeichen	E			
Nr. 070	Zeichen	F			
Nr. 071	Zeichen	G			
Nr. 072	Zeichen	H			
Nr. 073	Zeichen	I			
Nr. 074	Zeichen	J			
Nr. 075	Zeichen	K			
Nr. 076	Zeichen	L			
Nr. 077	Zeichen	M			
Nr. 078	Zeichen	N			
Nr. 079	Zeichen	O			
Nr. 080	Zeichen	P			
Nr. 081	Zeichen	Q			
Nr. 082	Zeichen	R			
Nr. 083	Zeichen	S			
Nr. 084	Zeichen	T			
Nr. 085	Zeichen	U			
Nr. 086	Zeichen	V			
Nr. 087	Zeichen	W			
Nr. 088	Zeichen	X			
Nr. 089	Zeichen	Y			
Nr. 090	Zeichen	Z			
Nr. 091	Zeichen	[
Nr. 092	Zeichen	\			
Nr. 093	Zeichen]			
Nr. 094	Zeichen	^			
Nr. 095	Zeichen	~			
Nr. 096	Zeichen				
Nr. 097	Zeichen	a			



	Datei	Bearbeiten	Format	Ansicht	?
Nr. 099	Zeichen	c			
Nr. 100	Zeichen	d			
Nr. 101	Zeichen	e			
Nr. 102	Zeichen	f			
Nr. 103	Zeichen	g			
Nr. 104	Zeichen	h			
Nr. 105	Zeichen	i			
Nr. 106	Zeichen	j			
Nr. 107	Zeichen	k			
Nr. 108	Zeichen	l			
Nr. 109	Zeichen	m			
Nr. 110	Zeichen	n			
Nr. 111	Zeichen	o			
Nr. 112	Zeichen	p			
Nr. 113	Zeichen	q			
Nr. 114	Zeichen	r			
Nr. 115	Zeichen	s			
Nr. 116	Zeichen	t			
Nr. 117	Zeichen	u			
Nr. 118	Zeichen	v			
Nr. 119	Zeichen	w			
Nr. 120	Zeichen	x			
Nr. 121	Zeichen	y			
Nr. 122	Zeichen	z			
Nr. 123	Zeichen	{			
Nr. 124	Zeichen				
Nr. 125	Zeichen	}			
Nr. 126	Zeichen	~			
Nr. 127	Zeichen	␣			
Nr. 128	Zeichen	€			
Nr. 129	Zeichen				
Nr. 130	Zeichen	f			
Nr. 131	Zeichen				

Verwendung der Datentypen

Einzelne Zeichen (char)

können auch zur Aufnahme von ganzzahligen Werten benutzt werden:

unsigned char ... kann 0 bis 255 repräsentieren (vorzeichenlos)

char ... kann Werte von -128 bis +127 repräsentieren

Ob eine char-Variable als Zeichen, oder als Zahl benutzt wird, hängt immer vom Kontext ab.

Benutzung als Zeichen:

```
char z; int i=0;
do {
    z = getchar(); // Eingabe Zeichen
    if (z!='x') i = i+1;
    else break;
} while (true);
```

Benutzung als Zahlenwert:

```
char z=0;
do {
    if (z<12) printf(" %d Uhr morgens\n",z);
    else if (z==12)
        printf(" %d Uhr mittags\n",z);
    else
        printf(" %d Uhr nachmittags\n",z-12);
    z = z + 1;
} while (z<24);
```

Konstantenausdrücke

Konstantenausdrücke sind

- ganze Zahlen, z.B. *123*, *-465*, *033*, *0xab*, *0XFF*, *123L*, *123UL*,
- Gleitkommazahlen, z.B. *12.34*, *12.45e-3*, *0123*, *1e20*,
- Zeichenkonstanten, z.B. *'a'*, *'X'* und
- Aufzählungswerte, z.B. *rot*, *Montag*, ...
(wenn vorher entsprechend definiert)

Konstante Zeichenfolgen sind z.B.

- *"Guten Morgen"* (besteht aus Ein-Byte-Zeichen) und
- *L"Guten Morgen"* (besteht aus Mehr-Byte-Zeichen)

Konstantenausdrücke

Konstantenausdrücke werden in Zuweisungen benutzt, z.B. für Anfangswerte

```
int startwert=5;
```

auch für 'feste' Werte in Berechnungen

```
float umfang = 2*a+2*b;
```

```
float flaeche= 0.433f*a*a;
```

Konstante Variablen

Variablen können als „const“ markiert werden. Dann darf der Wert einer Variable später nicht mehr geändert werden.

Die Verwendung von Konstanten erhöht die Lesbarkeit des Programms und macht es änderungsfreundlicher

Anstatt: *umfang = 2 * radius * 3.14159;*

Besser: *const double PI = 3.14159;*

*umfang = 2 * radius * PI;*

const heißt nur, dass die Variable nicht mehr verändert werden darf, der Wert muss nicht schon zur Übersetzungs-Zeit bestimmt werden können.

Beispiel:

*const double UmfangMeinKreis = 2.0 * radius * PI;*

// radius muss keine Konstante sein

Konstante Variablen

Guter Programmierstil ist es, außer den Konstanten -1 , 0 und 1 keine numerischen Konstantenausdrücke in einem Programm zu verwenden, sondern diese immer einer konstanten Variable zuzuweisen.

Anstatt:

```
for (int i=0;i<10;i++)  
    spieler[i].anzahl_huetchen= ...
```

Besser:

```
const int ANZAHL_MITSPIELER = 10;  
...  
for (int i=0; i < ANZAHL_MITSPIELER; i++)  
    spieler[i].anzahl_huetchen=...
```

Aufzählungstypen

Wenn eine Auswahl möglicher (nichtnumerischer) Werte abgebildet werden muss, bietet sich ein Aufzählungsdatentyp an.

Beispiel:

```
enum Wochentag { Mon, Die, Mit, Don, Fri, Sam, Son };  
Wochentag Tag;  
Tag = Mon;
```

Die Variable Tag kann nur Werte annehmen, die der bei der Deklaration von Wochentag angegeben wurden.

Syntax von Aufzählungen:

```
enum AufzTyp { Bezeichner1, Bezeichner2, ... } Variable;  
enum { Bezeichner1, Bezeichner2, ... } Variable;
```

Interne Informationsdarstellung

Interne Darstellung

- ganzzahliger Datentypen
- Fließkomma-Datentypen

Was passiert:

- Dezimalkonstantenausdrücke werden vom Übersetzer in eine interne Binärdarstellung umgewandelt
- Eingeegebene Werte (z.B. mit scanf) werden durch die Eingabefunktion in eine Binärdarstellung umgewandelt
- Rechenoperationen werden intern immer in der Binärdarstellung ausgeführt
- Ausgabefunktionen (z.B. printf) wandeln die binären Repräsentation der Variablen wieder in dezimale Werte zurück.

Ganzzahlige Datentypen

In C werden ganzzahlige Datentypen in den Kategorien ohne Vorzeichen (**unsigned**) und mit Vorzeichen (**signed**) unterstützt.

16-Bit: *unsigned short*

Größe (rein binär)

Bit 15 14 ... ← ... 2 1 0

Die Größe errechnet sich aus den zugehörigen Zweierpotenzen:

$$b_{15}2^{15} + b_{14}2^{14} + \dots + b_22^2 + b_12^1 + b_02^0$$

Die b_i stellen die betreffenden Bitwerte an der i-ten Stelle dar.

$$USHRT_MAX = 65535 = 2^{16} - 1$$

Ganzzahlige Datentypen

32-Bit: *unsigned int*

Größe (rein binär)

Bit 31 30 ... ← ... 2 1 0

Ähnlich wie 16-Bit Integer-Zahl, nur mit mehr Bitstellen und damit einem größeren Darstellungsbereich. Der gespeicherte Zahlenwert errechnet sich aus den zugehörigen Zweierpotenzen:

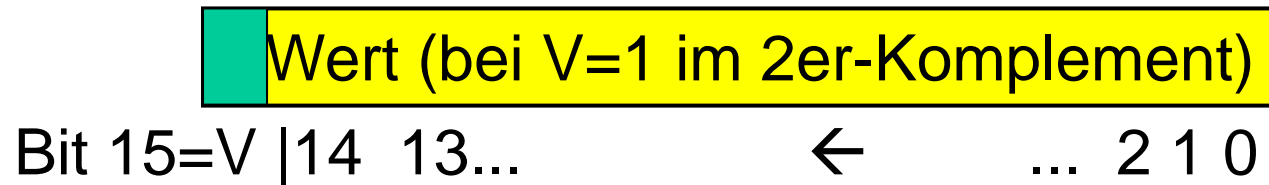
$$b_{31}2^{31} + b_{30}2^{30} + \dots + b_22^2 + b_12^1 + b_02^0$$

Die b_i stellen die betreffenden Bitwerte an der i-ten Stelle dar.

$$\text{UINT_MAX} = 4294967295 = 2^{32} - 1$$

Ganzzahlige Datentypen

16-Bit: *signed short*



Das werthöchste Bit (hier 15) ist das Vorzeichen-Bit V.

V=1 bedeutet eine negative Zahl

V=0 bedeutet eine positive Zahl

Die Größe errechnet sich *bei positiven Zahlen* aus den zugehörigen Zweierpotenzen:

$$b_{14}2^{14} + b_{13}2^{13} + \dots + b_22^2 + b_12^1 + b_02^0$$

Bei *negativen Zahlen* sind die b_i im sogenannten *Zweier-Komplement* dargestellt.

Negative Zahlen im Zweierkomplement

Das Zweierkomplement einer Zahl erhält man, indem man die Binärdarstellung der Zahl bitweise komplementiert (Einer-Komplement) und danach eine „1“ addiert.

Beispiel: Zahl = -1

Binärdarstellung: 0000 0000 0000 0001

Einerkomplement: 1111 1111 1111 1110

1 - Addition: 1111 1111 1111 1111

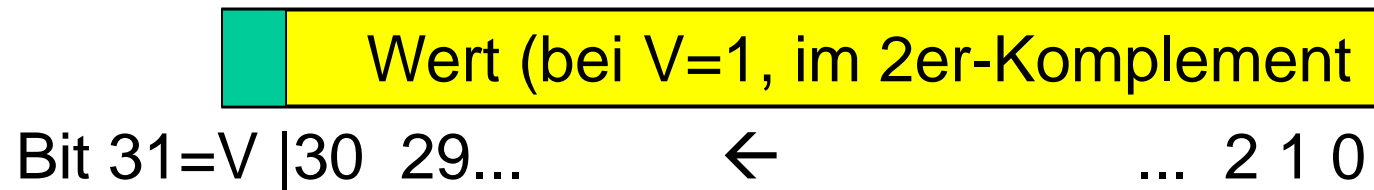
Diese Darstellung ist die -1!

Man sieht, das durch die Komplementierung immer das V-Bit auf „1“ gesetzt ist !

SHRT_MIN=-32768 SHRT_MAX=32767

Ganzzahlige Datentypen

32-Bit: *signed int*



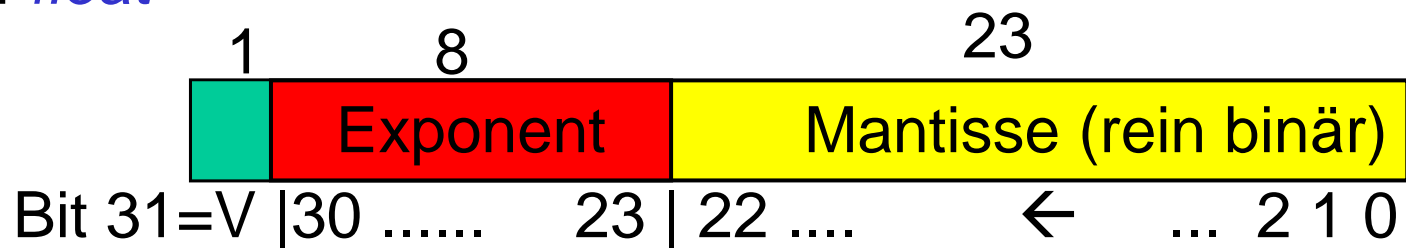
Das werthöchste Bit (hier 31) ist das Vorzeichen-Bit V.

INT_MIN=-2147483648 INT_MAX=2147483647

Gleitkomma-Datentypen

In C werden Gleitkomma-Datentypen in drei verschiedenen Genauigkeits-Kategorien (*float*, *double*, *long double*) unterstützt.

32-Bit: *float*



Das Vorzeichenbit V(Bit 31) bezieht sich nur auf den Wertanteil der Zahl. Der Wert der Mantisse *m* errechnet sich aus den zugehörigen **negativen** Zweierpotenzen:

$$m = b_{22}2^{-1} + b_{21}2^{-2} + \dots + b_22^{-21} + b_12^{-22} + b_02^{-23}$$

Die b_i stellen die betreffenden Bitwerte an der i-ten Stelle dar.

Gleitkomma Datentypen: float

Zum Wert der Mantisse m wird eine 1 addiert. Den Wert $1 + m$ nennt man **Signifikand**. Die 1 ist also nicht mit abgespeichert, sondern nur implizit vorhanden.

Der **Exponent** ist vorzeichenbehaftet und zur **Basis 2**. Dies wird erreicht, indem man den Exponenten um eine feste Größe (**BIAS**) erhöht darstellt. Der BIAS berechnet sich aus der Anzahl der Bits n des Exponenten:

$$\text{BIAS} = 2^{n-1} - 1$$

Mit $n=8$ ergibt sich 127.

Demnach muss der Binärwert des Exponenten e (die Bitstelle 23 hat dabei den Stellenwert 2^0) um den BIAS vermindert werden, um den realen Wert des Exponenten zur Basis 2 zu erhalten.

Gleitkomma Datentypen: float

Der Wert Z einer 32-Bit float-Zahl ist damit

$$Z = V \text{ Signifikand} * 2^{e - \text{BIAS}}$$

Das Vorzeichen V ist $+$, falls der Bitwert von $b_{31}=0$ ist, sonst $-$

Beispiel:

V Exponent

Mantisse

1 1000 0001 0100 0000 0000 0000 0000 000

- $e = 129$ $m = 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + \dots + 0*2^{-23}$

- $e - \text{BIAS} = 2$ $m = 1*0.25 = 0.25$

$$Z = - (1 + 0.25) * 2^2 = - 1.25 * 4 = -5$$

Gleitkomma Datentypen: float

Damit ergeben sich für 32-Bit float-Zahlen folgende Darstellungsbereiche:

FLT_MAX=3.40282e+38 (in dieser Darstellung ist e zur Basis 10 !)

FLT_MIN=1.17549e-38

Durch die begrenzte Anzahl von Bits zur Darstellung der Mantisse, ist die Anzahl genau darstellbarer Ziffern in einer Dezimaldarstellung ebenfalls begrenzt. Dabei werden die niederwertigen Ziffern betroffen.

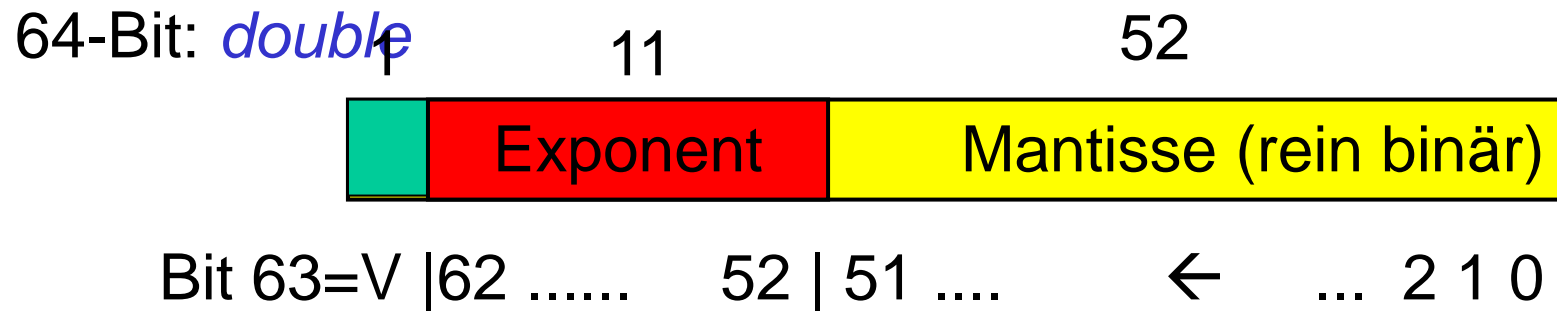
Anzahl der genau dargestellten Dezimalziffern: 6

Das heißt:

100100.8 und 100100.9 nicht mehr sicher unterscheidbar, da niederwertigste Ziffer sich an Position 7 befindet.

Bei 20 011 292 als Geldbetrag wären die Zehner- und Einerstelle nicht genau abgebildet.

Interne Informationsdarstellung



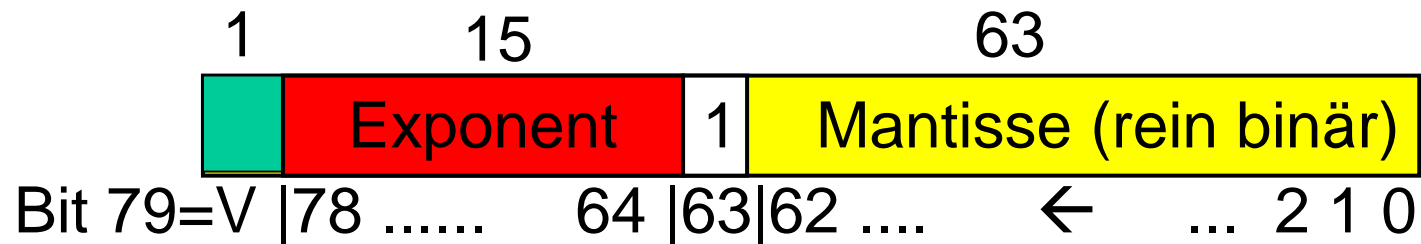
Damit ergeben sich für 64-Bit double-Zahlen folgende Darstellungsbereiche:

$DBL_MAX=1.797693e+308$ (in dieser Darstellung ist
 $DBL_MIN=2.225074e-308$ e zur Basis 10 !)

Anzahl der genau dargestellten Dezimalziffern: 15

Interne Informationsdarstellung

80-Bit: *long double*



Das Bit 63=1 nennt sich Integerbit und wird nur bei *long double* benötigt.

Für **80-Bit** *long double*-Zahlen ergeben sich folgende Darstellungsbereiche:

LDBL_MAX ~ 1.1e+4932 (in dieser Darstellung ist

LDBL_MIN ~ 3.4e-4932 e zur Basis 10 !)

Anzahl der genau dargestellten Dezimalziffern: 19

Interne Informationsdarstellung

Die Konstanten

FLT_MIN, FLT_MAX, DBL_MIN, DBL_MAX,
LDBL_MIN, LDBL_MAX

werden durch `#include<float.h>` verfügbar.

Weitere Konstanten aus `float.h`:

- FLT_EPSILON ... kleinster float-Wert x für den $1.0+x \neq 1.0$ gilt
- DBL_EPSILON und LDBL_EPSILON dementsprechend für float und long float.
- FLT_DIG ... Anzahl genau dargestellter Ziffern bei float Zahl (6)
- DBL_DIG und LDBL_DIG dementsprechend (15, bzw. 19)