Codes zur Darstellung von Zeichen
ASCII-Code

Der ASCII-Code (American Standard for Coded Information Interchange) ist eine festgelegte Abbildungsvorschrift (Norm) zur binären Kodierung von Zeichen.

- Der ASCII-Code umfasst Klein-/Großbuchstaben des lateinischen Alphabets, (arabische) Ziffern und viele Sonderzeichen.
- Die Kodierung erfolgt in einem Byte (8 Bits), so dass mit dem ASCII-Code 256 verschiedene Zeichen dargestellt werden können.
- Da das erste Bit nicht vom Standard-ASCII-Code genutzt wird, können im Standard-ASCII-Code nur 128 Zeichen dargestellt werden. Unterschiedliche, speziell normierte, ASCII-Code-Erweiterungen nutzen das erste Bit, um weitere 128 Zeichen darstellen zu können.

Codes zur Darstellung von Zeichen

ASCII-Code (Beispiel und Ausschnitt)

ASCII-Code zu den darstellbaren Zeichen

Zeichen	Dez.	Binär	Hexa	Oktal	Zeichen	Dez.	Binär	Hexa	Oktal
!	33	0010 0001	21	041	P	80	0101 0000	50	120
"	34	0010 0010	22	042	Q	81	0101 0001	51	121
#	35	0010 0011	23	043	R	82	0101 0010	52	122
$	36	0010 0100	24	044	S	83	0101 0011	53	123
%	37	0010 0101	25	045	T	84	0101 0100	54	124
&	38	0010 0110	26	046	U	85	0101 0101	55	125
'	39	0010 0111	27	047	V	86	0101 0110	56	126
(	40	0010 1000	28	050	W	87	0101 0111	57	127
)	41	0010 1001	29	051	X	88	0101 1000	58	130

Codes zur Darstellung von Zeichen
ASCII-Code

Zur Speicherung von Texten werden einzelne Bytes, die jeweils immer ein Zeichen kodieren, einfach hintereinander abgespeichert, so dass man eine Zeichenkette (*String*) erhält.

Um das Ende der Zeichenkette zu identifizieren, werden (in den Programmiersprachen) unterschiedliche Verfahren verwendet.

- Die Länge der Zeichenkette wird im ersten bzw. in den ersten Bytes vor der eigentlichen Zeichenkette gespeichert. Dieses Verfahren benutzt z. B. die Programmiersprache PASCAL.
- Das Ende der Zeichenkette wird durch ein besonderes, nicht darzustellendes Zeichen gekennzeichnet. So verwendet z.B. die Programmiersprache C/C++ ein 0-Byte (Byte, in dem alle Bits 0 sind), um das Ende einer Zeichenkette zu kennzeichnen.

Codes zur Darstellung von Zeichen
ASCII-Code

Unterscheidung zwischen Ziffern und Zeichen

- *Ziffer als ASCII-Code $\longrightarrow$ Angabe des Zeichens (Ziffer) in Hochkomma:*

'0': 00110000 (Dezimal 48)
'4': 00110100 (Dezimal 52)
'5': 00110101 (Dezimal 53)
'8': 00111000 (Dezimal 56)

- *Ziffer als numerischer Wert $\longrightarrow$ Angabe einer Ziffer (ohne Hochkomma):*

0: 00000000 (Dezimal 0)
4: 00000100 (Dezimal 4)
5: 00000101 (Dezimal 5)
8: 00001000 (Dezimal 8)

Codes zur Darstellung von Zeichen

ASCII-Code

Beispiele zum Speichern von Zeichen im ASCII-Code:

Angabe	Dezimaler ASCII–Wert	Dualdarstellung im Rechner
'a'	97	01100001
'W'	87	01010111
'*'	42	00101010
'9'	57	00111001

Codes zur Darstellung von Zeichen

Uni-Code

Der ASCII-Code mit seinen 256 Zeichen ist doch sehr begrenzt. Mit dem Unicode wurde ein Code eingeführt, in dem die Zeichen oder Elemente praktisch aller bekannten Schriftkulturen und Zeichensysteme festgehalten werden können.

Die Zeichenwerte der von Unicode erfassten Zeichen wurden bis vor kurzem noch ausschließlich durch eine zwei Byte lange Zahl ausgedrückt. Auf diese Weise lassen sich bis zu 65 536 verschiedene Zeichen in dem System unterbringen
(2 Byte = 16 Bit = 2^{16} Kombinationsmöglichkeiten).

In der Version 3.1 wurden 94 140 Zeichen aufgenommen, wobei die Zwei-Byte-Grenze durchbrochen wurde. Das Zwei-Byte-Schema, im Unicode-System als *Basic Multilingual Plane (BMP)* bezeichnet, wird deshalb von einem Vier-Byte-Schema abgelöst.

Weitere Codes
BCD-Code

BCD-Werte (Binary Coded Decimals) sind eine weitere Art der binären Kodierung von Zahlen bzw. Ziffern.

Für jede Dezimalziffer werden mindestens vier, manchmal auch acht Bits verwendet. Die jeweiligen Ziffern werden nacheinander immer durch ihren Dualwert angegeben.
Es ist eigentlich eine Speicherplatz verschwendende Art der Speicherung von Dezimalzahlen, erleichert aber manche Anwendungen.

Anwendungsbereiche:

- Rechnen im Dezimalsystem,
- Speichern von Dezimalzahlen (Telefonnummern u.ä.),
- Ansteuerung von LCD-Anzeigen, um Dezimalziffern einzeln anzuzeigen.

Weitere Codes
BCD-Code

- Beispiele:

Dezimalzahl	Dualzahl	Duale BCD-Darstellung
294	100100110	0010.1001.0100 2 9 4
16289	11111110100001	0001.0110.0010.1000.1001 1 6 2 8 9

- Die Bitmuster 1010, 1011, …, 1111 werden im BCD-Code nicht für Ziffern benötigt. Sie können genutzt werden z.B. für Vorzeichen +: 1010 und -: 1011

Weitere Codes

Gray-Code

Beim Gray-Code zur Kodierung von Binärzahlen unterscheiden sich zwei aufeinanderfolgende Codewörter immer nur um genau ein Bit.

Dezimal	Gray (Binär)	Dezimal	Gray (Binär)	Dezimal	Gray (Binär)
1	0001	6	0101	11	1110
2	0011	7	0100	12	1010
3	0010	8	1100	13	1011
4	0110	9	1101	14	1001
5	0111	10	1111	15	1000

Weitere Codes
Gray-Code

Verwendung findet der Code z.B. für die binäre Ausgabe von Werten von A/D-Wandlern (A/D = Analog/Digital) zur Vermeidung unsinniger Zwischenwerte beim Auslesen.

Da sich bei jedem Zahlenübergang immer jeweils nur ein Bit ändert, werden unsinnige Zwischenwerte bei Übergängen von z.B. (7) 0111 zu (8) 1000 vermieden (z.B. 1111(15)), wenn die Übergänge von 0→1 und 1→0 unterschiedlich schnell in der HW ablaufen.

Im Gray-Code können Werte gespeichert werden.
Sollen Werte in Gray-Zahlen arithmetisch weiterverarbeitet werden, müssen diese dazu natürlich erst in Dualzahlen umgewandelt werden.

Duale Größenangaben

Maßeinheiten für Bytes

Maßeinheit		Anzahl von Bytes	KBytes	MBytes
Byte		1		
Kilobyte (KByte)	2^{10}	1024	1	
Megabyte (MByte)	2^{20}	1.048.576	1024	1
Gigabyte (GByte)	2^{30}	1.073.741.824	1.048.576	1024
Terabyte (TByte)	2^{40}	1.099.511.627.776	1.073.741.824	1.048.576
Petabyte (PByte)	2^{50}	1.125.899.906.842.624	1.099.511.627.776	1.073.741.824
Exabyte (EByte)	2^{60}	1.152.921.504.606.846.976	1.125.899.906.842.624	1.099.511.627.776

Grunddatentypen

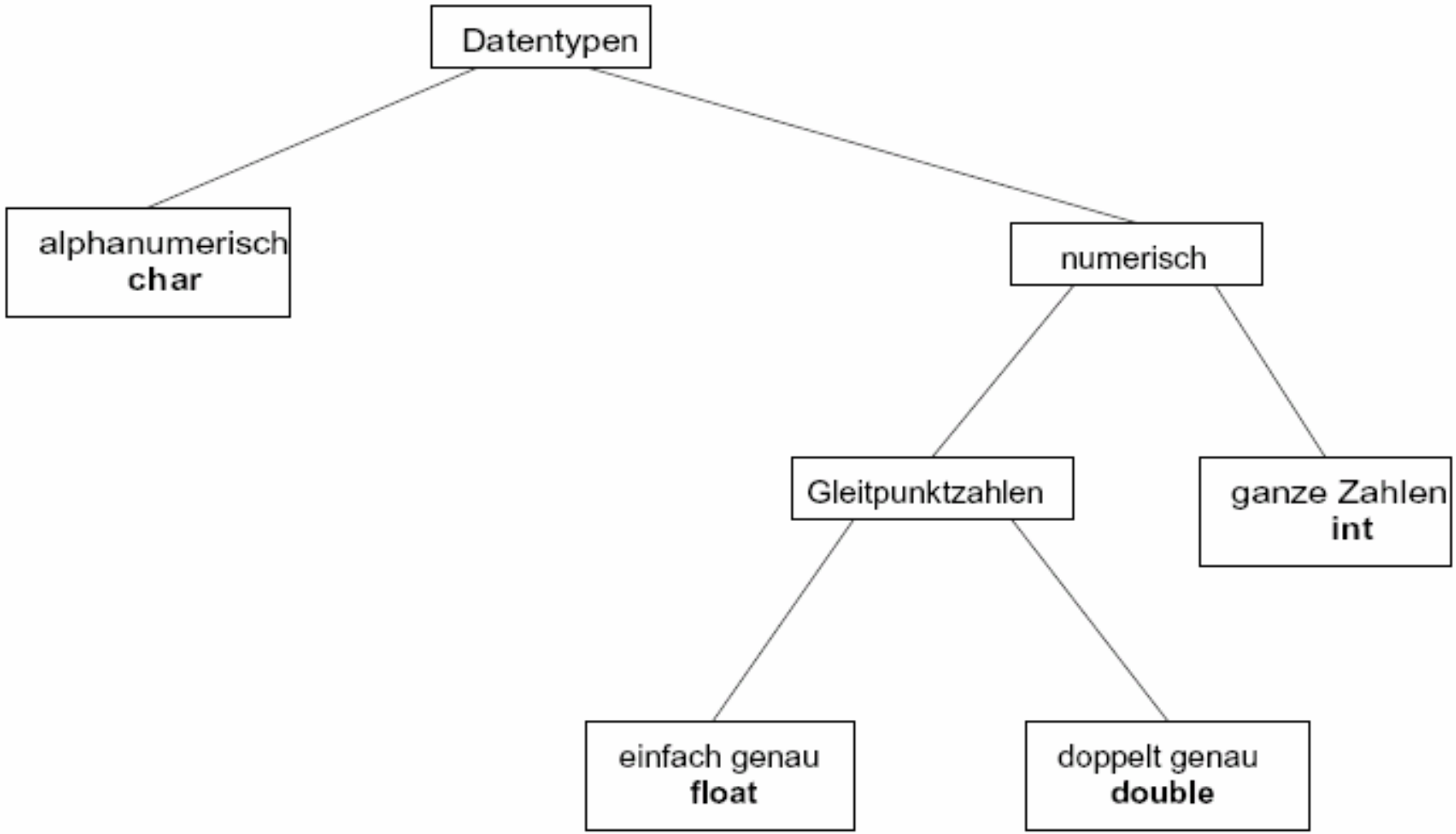
In einem Computer werden Zeichen – wie z. B. Buchstaben – anders behandelt als ganze Zahlen und Gleitpunktzahlen wie z.B. die Zahl $\pi = 3.1415\ldots$

Daher ist eine Klassifikation dieser unterschiedlichen Daten notwendig. Die unterschiedlichen Datentypen unterscheiden sich einerseits im Speicherbedarf und damit der darstellbaren Größe von Zahlen bzw. des Zeichenvorrats, und andererseits in der Interpretation des gegebenen Bitmusters.

Ordnet man in einem Programm Daten bestimmten Klassen wie *Zeichen, ganze Zahl, einfach/doppelt genaue Gleitpunktzahl* usw. zu, dann teilt man dem Rechner deren *Datentyp* mit.

Grunddatentypen

C-Grunddatentypen



Grunddatentypen und Wertebereiche

C-Datentypen auf 32-Bit-Architekturen

Datentyp-Bezeichnung	Bitzahl	Wertebereich
`char, signed char`	8	−128...127
`unsigned char`	8	0...255
`short, signed short`	16	−32 768...32 767
`unsigned short`	16	0...65 535
`int, signed int`	32	−2 147 483 648...2 147 483 647
`unsigned, unsigned int`	32	0...4 294 967 295
`long, signed long`	32	−2 147 83 648...2 147 483 647
`unsigned long`	32	0...4 294 967 295
`float`	32	$1.2 \cdot 10^{-38} \ldots 3.4 \cdot 10^{38}$
`double`	64	$2.2 \cdot 10^{-308} \ldots 1.8 \cdot 10^{308}$

Grunddatentypen

Verlust von Bits bei zu großen Zahlen

Wird versucht, in einem Datentyp einen Wert abzulegen, der nicht in diesen Datentyp passt, so werden einfach die vorne überhängenden Dualziffern abgeschnitten.

Es soll versucht werden, die Zahl `50` im hypothetischen 4-Bit Datentyp `kurz` darzustellen:

$$(50)_{10} = (110010)_2$$

Da nur für vier Bits Platz im Datentyp `kurz` ist, werden die ersten beiden Bits (`11`) einfach weggeworfen:

11|0010 , so dass schließlich folgende Bitkombination in `kurz` gespeichert wird:

0010

Dies ist die Dualdarstellung für die Zahl `2`. Der Versuch, die Zahl `50` im Datentyp `kurz` unterzubringen, führte also schließlich dazu, dass dort die Zahl `2` gespeichert wurde.

Grunddatentypen

Verlust von Bits bei zu großen Zahlen

Es soll versucht werden, die Zahl 43 im hypothetischen 4-Bit Datentyp kurz darzustellen:

$$(43)_{10} = (101011)_2$$

Da nur für 4 Bits Platz im Datentyp kurz ist, gehen die ersten beiden Bits (10) einfach wieder verloren: 10|1011

so dass sich schließlich folgende Bitkombination in kurz ergibt:

1011

Dies ist die Dualdarstellung für die Zahl -5. Der Versuch, die Zahl 43 im Datentyp kurz unterzubringen, führte also schließlich dazu, dass dort die Zahl -5 gespeichert wurde.

Vorsicht:

In Programmiersprachen wie C/C++ und auch Java wird beim Abspeichern von Zahlen, die außerhalb des Wertebereichs eines Datentyps liegen, meist *kein* Fehler gemeldet, sondern es werden die überhängenden Bits abgeschnitten! Mit diesem falschen Wert wird weiter gearbeitet, was schließlich zu falschen Ergebnissen führt.