Gliederung des Kapitels 1.1

1.1 Algorithmen und Sprachen

~~1.1.1 Darstellung von Algorithmen~~
1.1.2 Grundlegende Datenbereiche
1.1.3 Realisierte Abbildung
1.1.4 (Künstliche) Sprachen
1.1.5 Grammatiken
1.1.6 BNF, Syntaxdiagramme
1.1.7 Sprachen zur Beschreibung von Sprachen
1.1.8 Übungsaufgaben

Abschnitt 1.1.2 soll Ihnen folgende Inhalte vermitteln:

Elementare Wertebereiche der Programmierung, also die Wertebereiche, die man also nicht auf andere Bereiche zurückführt.

Für jeden dieser Bereiche muss eindeutig festgelegt sein, wie seine Elemente dargestellt (aufgeschrieben oder im Rechner repräsentiert) sind. Sie lernen diese Darstellungen kennen.

Zugleich stellen wir alternative Darstellungen vor. An diesen erkennen Sie einige Vor- und Nachteile von Darstellungen; diese hängen meist von den späteren Anwendungen ab, so dass Sie ein Gespür dafür erhalten sollen, in welchem Fall welche Darstellung nutzbringend ist.

Weiterhin lernen Sie einige Phänomene (wie z.B. Rundungsfehler) und mehrere typische Beispiele. Im Vorgriff auf später führen wir am Ende Unterstrukturen und arrays ein.

1.1.2 Grundlegende Datenbereiche

Programme verändern Daten. Diese Daten werden in (Informatik-) Variablen abgelegt. Eine solche Variable ist ein Behälter, wobei vorher festgelegt wird, welche Werte in den Behälter gelegt werden dürfen und welche nicht.

Durch diese *Festlegung der zulässigen Wertemenge* in den Deklarationen erhält eine Variable einen "Typ", den sog. Datentyp. Wenn eine Variable z.B. den Datentyp "integer" erhält, so darf sie nur ganze Zahlen als Werte enthalten, erhält sie den Typ "character", so darf sie nur Zeichen des Alphabets enthalten usw.

Bei den Datentypen ist es wie bei den Anweisungen: Es gibt "elementare Datentypen" (diese gehören zu Wertebereichen, die als grundlegend angesehen werden) und es gibt Regeln, wie man aus bereits definierten Datentypen neue Datentypen gewinnt.

Was sind "elementare Datentypen", also Wertebereiche, die man nicht aus noch einfacheren Mengen zusammensetzen kann oder will?

In den meisten Programmiersprachen zählt man hierzu die Menge der Wahrheitswerte, die Menge der Zeichen auf der Tastatur (Alphabetzeichen), die natürlichen, die ganzen und die reellen Zahlen sowie die endlichen Mengen, die man selbst durch Auflisten ihrer Elemente definiert.

Einschub: Was Sie in der Mathematik lernten:

Eine Menge ist eine Zusammenfassung von paarweise verschiedenen Elementen zu einem Ganzen.

Eine Menge kennzeichnet man stets durch geschweifte Klammern "{" und "}".
Man kann sie beschreiben, indem man ihre Elemente auflistet, evtl. mit "...", sofern das Bildungsgesetz klar ist, z.B.:

Dezimalziffer = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9},
$\mathbb{B}$ = {0, 1} (Menge der Binärziffern bzw. der Booleschen Werte),
Wochentag = {Mo, Di, Mi, Do, Fr, Sa, So},
$\mathbb{N}_0$= {0, 1, 2, 3, ...} (Menge der natürlichen Zahlen mit der 0).

Noch Einschub

Oder man kann eine Menge M durch eine charakteristische Eigenschaft beschreiben nach dem Schema

M = { a | a besitzt die Eigenschaft .... }.

Beispiele:
G = {a | a ist eine Dezimalziffer und a ist als Zahl durch 2 teilbar},
T = {w | w ist ein Familienname und w kommt im Telefonbuch von Stuttgart des Jahres 2002 vor},
Prim = {p | p ist eine natürliche Zahl, p > 1 und p ist nur durch die Zahlen 1 und p teilbar},
E = {w | w ist der Name eines Ortes, dessen kürzeste Entfernung nach Stuttgart zwischen 107 und 117 km beträgt}.

Noch Einschub

Wenn M eine Menge ist und a zur Menge M gehört, so sagt man, a ist ein Element von M, und schreibt hierfür: $a \in M$. Gehört a nicht zu M, so schreibt man $a \notin M$.

In einer Menge kommt kein Element mehrfach vor. Die Reihenfolge, in der die Elemente aufgeschrieben werden, spielt keine Rolle. Die drei Mengen {1, 3, 1, 2, 3, 3, 3}, {1, 2, 3} und {3, 1, 2} sind also gleich.

Gleichheit zweier Mengen: M = N genau dann, wenn

(1) für jedes $a \in M$ gilt $a \in N$ und
(2) für jedes $a \in N$ gilt $a \in M$.

Wenn M nicht gleich N ist, dann ist M ungleich N, geschrieben $M \neq N$. $M \neq N$ gilt also genau dann, wenn es ein Element a gibt, das in N, aber nicht in M, oder das in M, aber nicht in N liegt.

Noch Einschub

Definition: "Enthalten", "Teilmenge":
M heißt Teilmenge von N, im Zeichen: $M \subset N$ oder $N \supset M$, genau dann, wenn für jedes $a \in M$ gilt $a \in N$.

Ist M nicht Teilmenge von N, so schreibt man $M \not\subset N$.

M und N sind genau dann gleich, wenn M in N und N in M enthalten sind.

Die spezielle Menge, die kein Element besitzt, nennt man die leere Menge und bezeichnet sie mit dem Symbol $\emptyset$. Sie ist Teilmenge jeder Menge.

Grundlegende Mengen für die Programmierung sind:
IB = {false, true} Boolesche Werte, Wahrheitswerte, oft: {0, 1}.
$\mathbb{N}$ = {1, 2, 3, 4, ...} ist die Menge der natürlichen Zahlen,
$\mathbb{N}_0$= {0, 1, 2, 3, ...} die Menge der natürlichen Zahlen mit der 0,
$\mathbb{Z}$ = { ..., -2, -1, 0, 1, 2, 3, ... } ist die Menge der ganzen Zahlen,
Â sei die Menge der Tastaturzeichen zuzüglich einiger sog. Steuerzeichen (vgl. später: ASCII-Code, EBCDIC-Code).
$\mathbb{Q}$ = {n/m | n ganze Zahl, m natürliche Zahl, n und m teilerfremd} ist die Menge der rationalen Zahlen.
$\mathbb{R}$ bezeichnet die Menge der reellen Zahlen (Vervollständigung der rationalen Zahlen, so dass jeder Punkt der Zahlengeraden (als Grenzwert einer Cauchyfolge) erfasst wird, z.B. die Kreiszahl π oder die Wurzel aus 2).
$\mathbb{C}$ ist die Menge der komplexen Zahlen; formal kommt "i" (= die Wurzel aus -1) zu $\mathbb{R}$ hinzu; siehe Mathematikvorlesungen.

Elementare Datentypen 1.1.2.1:
Wenn eine Variable nur Werte aus der Menge M besitzen darf, so geben wir ihr in einer Deklaration folgenden Datentyp:

Menge	Datentyp
IB	Boolean oder logical
$\mathbb{N}_0$	natural oder Cardinal
$\mathbb{Z}$	integer
$\mathbb{R}$	real oder float
Â	character

$\mathbb{N}$ betrachtet man nicht gesondert, sondern man verwendet hierfür "natural" und stellt im Programm durch Bedingungen sicher, dass der Wert 0 nicht auftritt.

$\mathbb{C}$ führt man meist selbst als Paar zweier reeller Zahlen ein und definiert für diese Paare die komplexen Operatoren Addition, Subtraktion, Multiplikation, Division usw., siehe späteres Kapitel 1.3.

Eine Sonderrolle spielen die rationalen Zahlen $\mathbb{Q}$; denn eigentlich beschreibt der Typ "real" in der Praxis nicht die ganze Menge der reellen Zahlen, sondern nur eine gewisse Teilmenge der rationalen Zahlen. Wir werden dies später unter dem Stichwort "Zahlendarstellungen" noch in diesem Abschnitt untersuchen, wenden uns jedoch zuerst Beispielen zu.

Beispiel 1.1.2.2: Fakultätsfunktion für natürliche Zahlen

Die Fakultät n! ist definiert als das Produkt der ersten n natürlichen Zahlen. Also: n! = 1·2· ... ·n. Zusätzlich definiert man 0! = 1.

Dies berechnet man mit einer for-Schleife:

```
ergebnis := 1;
for i := 1 to n do ergebnis := ergebnis*i od.
```

Den Variablen ergebnis und i geben wir den Datentyp natural und erhalten somit das Programm

```
program fakultaet1 is
var ergebnis, i, n: natural;
begin
        read (n);
        ergebnis := 1;
        for i := 1 to n do ergebnis := ergebnis*i od;
        write (ergebnis)
end
```


Beispiel 1.1.2.3: ggT (oder gcd)
größter gemeinsamer Teiler zweier natürlicher Zahlen
(im Englischen: gcd = greatest common divisor)

Sei $n \in \mathbb{N}_0$ eine natürliche Zahl. Sei
$T(n) = \{d \in \mathbb{N}_0 | d \text{ teilt } n, \text{ d.h., es gibt ein } c \in \mathbb{N}_0 \text{ mit } c \cdot d = n\}$
die Menge der Teiler von n.

Wegen $1 \in T(n)$ und $n \in T(n)$ ist T(n) niemals die leere Menge.
Für $n>0$ gilt: Wenn $d \in T(n)$ ist, so ist $d \leq n$. Speziell: $T(0) = \mathbb{N}_0$.

Bilde zu zwei Zahlen $a, b \in \mathbb{N}_0$ den Durchschnitt $T(a) \cap T(b)$.
Diese Menge ist nicht leer, da sie mindestens die Zahl 1
enthält. Das maximale Element in diesem Durchschnitt heißt
der größte gemeinsame Teiler von a und b, bezeichnet als
ggT(a,b). Außer für a=b=0 ist er stets eindeutig bestimmt.

Für $a, b, d \in \mathbb{N}_0$ mit $a \neq 0$ oder $b \neq 0$ gilt also $ggT(a,b) = d$ genau dann, wenn d sowohl a als auch b teilt und wenn für alle natürlichen Zahlen d', die sowohl a als auch b teilen, stets $d' \leq d$ gilt.

Zu a und b kann man ggT(a,b) daher berechnen, indem man alle Teiler von a und alle Teiler von b berechnet und hieraus die maximale Zahl ermittelt, die in beiden Teilermengen vorkommt. Dieses Verfahren dauert aber zu lange.

Man erhält ein schnelleres Verfahren, wenn man Eigenschaften ausnutzt, die der ggT(a,b) besitzen muss.

1.1.2.4. Eigenschaften des ggT:

(1)	$ggT(a,b) = ggT(b,a)$	für alle $a,b \in \mathbb{N}_0$, "Symmetrie",
(2)	$ggT(a,0) = ggT(a,a) = a$	für alle $a \in \mathbb{N}_0$,
(3)	$ggT(a,b) = ggT(a-b, b)$	für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$,
	$ggT(a,b) = ggT(a+b, b)$	für alle $a,b \in \mathbb{N}_0$,
(4)	$ggT(a,b) = ggT(b, a \bmod b)$	für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$.

Beweise: zu (1) und (2): Diese ersten beiden Zeilen folgen direkt aus der Definition und der Tatsache $T(0) = \mathbb{N}_0$.

zu (3): Wenn eine Zahl t sowohl a als auch b teilt, dann gibt es zwei Zahlen c_1 und c_2 mit $c_1 \cdot t = a$ und $c_2 \cdot t = b$. Hieraus folgt $a-b = c_1 \cdot t - c_2 \cdot t = (c_1 - c_2) \cdot t$, d.h., t teilt auch a-b. Folglich teilt der ggT(a,b) auch die Zahl a-b (für $c_1 - c_2 \geq 0$, d.h., für $a \geq b$). Offenbar teilt t stets auch a+b (ersetze einfach "-" durch "+").

Eigenschaften des ggT:

(1) $ggT(a,b) = ggT(b,a)$ für alle $a,b \in \mathbb{N}_0$, "Symmetrie",

(2) $ggT(a,0) = ggT(a,a) = a$ für alle $a \in \mathbb{N}_0$,

(3) $ggT(a,b) = ggT(a-b,b)$ für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$,
$ggT(a,b) = ggT(a+b,b)$ für alle $a,b \in \mathbb{N}_0$,

(4) $ggT(a,b) = ggT(b, a \bmod b)$ für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$.

Beweis zu (3), Fortsetzung:
Es sei $d = ggT(a,b)$. Wenn $d' = ggT(a-b,b)$ ist, so teilt d' die Zahl b und nach dem soeben Bewiesenen ist d' auch ein Teiler von $(a-b)+b = a$. Weil d der $ggT(a,b)$ ist, muss daher $d' \leq d$ sein. Andererseits ist d (wie jeder Teiler von a und b) auch Teiler von b und a-b, woraus $d \leq d'$ folgt, da d' der $ggT(a,a-b)$ ist. Aus beiden Ungleichungen folgt $d'=d$, d.h., $ggT(a,b) = ggT(a-b,b)$. Analog folgert man $ggT(a,b) = ggT(a+b,b)$.

Eigenschaften des ggT:

(1) $ggT(a,b) = ggT(b,a)$ für alle $a,b \in \mathbb{N}_0$, "Symmetrie",

(2) $ggT(a,0) = ggT(a,a) = a$ für alle $a \in \mathbb{N}_0$,

(3) $ggT(a,b) = ggT(a-b,b)$ für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$,
$ggT(a,b) = ggT(a+b,b)$ für alle $a,b \in \mathbb{N}_0$,

(4) $ggT(a,b) = ggT(b, a \bmod b)$ für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$.

Beweis zu (4):
Iteriere die Gleichung (3), woraus (4) mit (1) unmittelbar folgt:
$ggT(a,b) = ggT(a-b,b)$ für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$,
$ggT(a-b,b) = ggT(a-2 \cdot b,b)$ für alle $a,b \in \mathbb{N}_0$ mit $a-b \geq b$,
$ggT(a-2 \cdot b,b) = ggT(a-3 \cdot b,b)$ für alle $a,b \in \mathbb{N}_0$ mit $a-2 \cdot b \geq b$
usw., bis man $a-(k-1) \cdot b \geq b > a-k \cdot b$ für $k = a \operatorname{div} b$ erreicht, also:
$ggT(a-(a \operatorname{div} b - 1) \cdot b, b) = ggT(a-(a \operatorname{div} b) \cdot b, b) = ggT(a \bmod b, b)$
für alle $a,b \in \mathbb{N}_0$ mit $a \geq b$.

Aus Eigenschaft (4) erhalten wir sofort einen Algorithmus zur Berechnung des ggT, wobei Eigenschaft (2) als Terminierungsbedingung dient. Halb umgangssprachliche Formulierung:

```
read(A); read(B);   % falls A=B=0 ist, das Verfahren abbrechen
if A < B then "vertausche die Werte von A und B" fi;
while B ≠ 0 do
        nächster Wert von B wird A mod B;
        nächster Wert von A wird der Wert des alten B
od;
write(A)
```

Dieser Algorithmus endet, da anfangs $A \geq B$ ist, $B \geq A \bmod B$ stets gilt und im Falle der Gleichheit die while-Schleife abbricht. Die Werte von A und B werden also in jedem Schleifendurchgang verringert. d.h., B wird irgendwann 0.

Um zwei Werte auszutauschen oder einen "alten Wert" zuzuweisen, benötigt man Variablen zur Zwischenspeicherung.

Man kann "vertausche die Werte von A und B" nicht einfach durch A:=B; B:=A realisieren, da dann zwar A den Wert von B erhält, aber anschließend auch B diesen Wert bekommt, da der alte Wert von A ja bereits überschrieben wurde.

Wir verwenden also eine weitere Variable H, die an dieser Stelle derzeit nicht benötigt wird, und setzen:

H := A; A := B; B := H

So gehen wir auch im Schleifenrumpf vor.

```
program  euklid1  is
var  A, B, H: natural;
begin  read (A); read (B);
  if  (A > 0) or (B > 0)  then
     if A < B then  H := A;  A := B;  B := H  fi;
     while B ≠ 0 do
          H := A mod B;  A := B;  B:= H  od
  fi;
  write (A)
end
```

Hinweis: Nur im Falle a = b = 0 wird der Wert 0 ausgegeben.

(Ende des Beispiels 1.1.2.3 ggT)

Festlegung 1.1.2.5: *Anordnung der elementaren Wertebereiche.*

In den gängigen Programmiersprachen sind alle elementaren Datentypen total angeordnet, d.h., für je zwei verschiedene Elemente a und b gilt entweder a < b oder b < a.

Bei Boolean setzt man false < true, bei ganzen und reellen Zahlen nutzt man die natürliche Anordnung auf der Zahlengeraden und bei Alphabetzeichen nimmt man die Anordnung ihrer Codierung (s.u.). Wir legen daher fest, dass die Werte unserer fünf elementaren Datentypen (gemäß 1.1.2.1) in der genannten Weise angeordnet sind.

Wir wollen nun die Darstellung und die Anordnung der Alphabetzeichen Â exakt festlegen.

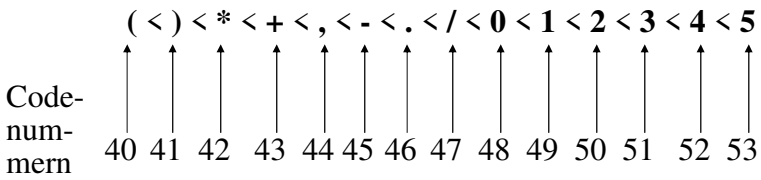
"Normale" Alphabetzeichen werden im Rechner durch 7 oder 8 Binärstellen ("Bits") dargestellt. Beispielsweise wird der Buchstabe A im 7-stelligen ASCII-Code als 1000001 beschrieben, der Buchstabe B als 1000010, C als 1000011 usw. Bereits in den 1950er Jahren wurde diese Darstellung vorgenommen. Der vollständige ASCII-Code steht auf den nächsten Folien.

Jede Folge von Nullen und Einsen kann als Binärdarstellung einer Zahl interpretiert werden. Im Falle eines siebenstelligen Codes $b_1b_2b_3b_4b_5b_6b_7$ lautet diese Zahl ("Codenummer" oder "Ordnungsnummer" genannt):
$64 \cdot b_1 + 32 \cdot b_2 + 16 \cdot b_3 + 8 \cdot b_4 + 4 \cdot b_5 + 2 \cdot b_6 + 1 \cdot b_7$.
Diese Zahl lautet somit 65 für A, 66 für B, 67 für C usw. Statt der dezimalen Darstellung 65, 66 usw. verwendet man in den Definitionstabellen oft Zahldarstellungen zur Basis 8 (Oktal-) oder 16 (Hexadezimaldarstellung).

1.1.2.6: ASCII-Code (ASCII ist die Abkürzung für "American Standard Code for Information Interchange").

Der Code in seiner alten Form umfasste $2^7 = 128$ Zeichen, von denen die 32 Zeichen für Zwecke der Steuerung im Computer verwendet wurden, 84 Zeichen waren fest mit Tastaturzeichen belegt und die verbleibenden 12 Zeichen konnten national genutzt werden (z.B. für Spezialzeichen wie ä, ö, ü, Ä, Ö, Ü, ß, [,]).

Die Anordnung der Alphabetzeichen erfolgte in der Reihenfolge der Codenummern jedes Zeichens. Beispielsweise gilt



ASCII-Code, erste Hälfte (nach ISO/IEC 646, vgl. DIN 66003)

Nr.	binär	Zeichen	Nr.	binär	Zeichen	Nr.	binär	Zeichen	Nr.	binär	Zeichen
0	0000000	NUL	16	0010000	DLE	32	0100000	Zwi.r	48	0110000	0
1	0000001	SOH	17	0010001	DC1	33	0100001	!	49	0110001	1
2	0000010	STX	18	0010010	DC2	34	0100010	"	50	0110010	2
3	0000011	ETX	19	0010011	DC3	35	0100011	#	51	0110011	3
4	0000100	EOT	20	0010100	DC4	36	0100100	$	52	0110100	4
5	0000101	ENQ	21	0010101	NAK	37	0100101	%	53	0110101	5
6	0000110	ACK	22	0010110	SYN	38	0100110	&	54	0110110	6
7	0000111	BEL	23	0010111	ETB	39	0100111	'	55	0110111	7
8	0001000	BS	24	0011000	CAN	40	0101000	(	56	0111000	8
9	0001001	HT	25	0011001	EM	41	0101001	)	57	0111001	9
10	0001010	LF	26	0011010	SUB	42	0101010	*	58	0111010	:
11	0001011	VT	27	0011011	ESC	43	0101011	+	59	0111011	;
12	0001100	FF	28	0011100	FS	44	0101100	,	60	0111100	<
13	0001101	CR	29	0011101	GS	45	0101101	-	61	0111101	=
14	0001110	SO	30	0011110	RS	46	0101110	.	62	0111110	>
15	0001111	SI	31	0011111	US	47	0101111	/	63	0111111	?

ASCII-Code, zweite Hälfte (nach ISO/IEC 646, vgl. DIN 66003)

Nr.	binär	Zeichen	Nr.	binär	Zeichen	Nr.	binär	Zeichen	Nr.	binär	Zeichen
64	1000000	@	80	1010000	P	96	1100000	`	112	1110000	p
65	1000001	A	81	1010001	Q	97	1100001	a	113	1110001	q
66	1000010	B	82	1010010	R	98	1100010	b	114	1110010	r
67	1000011	C	83	1010011	S	99	1100011	c	115	1110011	s
68	1000100	D	84	1010100	T	100	1100100	d	116	1110100	t
69	1000101	E	85	1010101	U	101	1100101	e	117	1110101	u
70	1000110	F	86	1010110	V	102	1100110	f	118	1110110	v
71	1000111	G	87	1010111	W	103	1100111	g	119	1110111	w
72	1001000	H	88	1011000	X	104	1101000	h	120	1111000	x
73	1001001	I	89	1011001	Y	105	1101001	i	121	1111001	y
74	1001010	J	90	1011010	Z	106	1101010	j	122	1111010	z
75	1001011	K	91	1011011	[	107	1101011	k	123	1111011	{
76	1001100	L	92	1011100	\	108	1101100	l	124	1111100	|
77	1001101	M	93	1011101	]	109	1101101	m	125	1111101	}
78	1001110	N	94	1011110	^	110	1101110	n	126	1111110	~
79	1001111	O	95	1011111	_	111	1101111	o	127	1111111	DEL

Meist schreibt man diesen Code als zweidimensionale Tabelle, siehe Literatur.

1.1.2.7: Codes mit 8 Binärstellen: EBCDIC-Code, Latein1.

Hat man eine Binärstelle mehr zur Verfügung, so lassen sich $2^8 = 256$ Zeichen darstellen. Der "extended binary coded decimal interchange code" EBCDIC verwendet ein Byte = 8 Binärstellen, von denen 64 Zeichen für die Steuerung reserviert sind. Die restlichen 192 Zeichen reichen aus, um alle westeuropäischen Schriftsprachen abzudecken (und damit auch alle in Amerika und Australien). Der Code "Latein1" ist eine Erweiterung der Codenummern 64 bis 127 des ASCII-Codes um 128 Zeichen des Dänischen, Spanischen, Tschechischen usw. Er kann leicht im EBCDIC beschrieben werden.

Zur genauen Definition und zu Erweiterungen des ASCII-Codes nach ISO 8859 siehe Literatur oder Vorlesungen der Technischen Informatik. In Ada kann mit Hilfe der Funktionen Val und Pos zwischen den Zeichen und ihren Codenummern hin und hergeschaltet werden.

Es fehlen noch arabische, kaukasische, chinesische, afrikanische usw. Schriftzeichen sowie die vielen Symbole und Sonderzeichen der verschiedenen Wissenschaften, insbesondere der Mathematik und der Naturwissenschaften.

Es gibt derzeit weniger als 30.000 solcher Zeichen. Um zugleich Platz für Erweiterungen zu haben, hat man den "Unicode" definiert, der aus 16 Binärstellen besteht, somit $2^{16} = 65536$ Zeichen beschreiben kann und alle bisher benutzten Zeichen einschl. der Hieroglyphen umfasst (siehe ISO-Standard 10646). In dessen ersten 256 Zeichen ist der ASCII-Code enthalten. Der Unicode besitzt noch Lücken, die erst künftig ausgefüllt werden.

Fazit: Man kann die Menge der Zeichen anordnen, speziell lassen sich die in Programmen verwendeten Tastaturzeichen anordnen, wobei deren Anordnung fast immer wie im ASCII-Code erfolgt.

Die Darstellung der natürlichen Zahlen ($\mathbb{N}_0$).

Natürliche Zahlen werden in der Regel durch ein Stellenwertsystem beschrieben.

Definition 1.1.2.8:
Ein Stellenwertsystem ist ein Tripel S = (b, Z, β) mit
(1) b ≥ 2 ist eine natürliche Zahl (die "Basis"),
(2) Z ist eine b-elementige Menge (die Menge der Ziffern),
(3) β: Z → {0, 1, ..., b-1} ⊂ $\mathbb{N}_0$ ordnet jedem Ziffernsymbol umkehrbar eindeutig eine Zahl zwischen 0 und b-1 zu.

Eine Zahl wird beschrieben durch eine Folge von Ziffern aus Z ohne führende Nullen (außer der Null selbst). Der Wert ϕ(z) einer n-stelligen Zahldarstellung $z = z_{n-1}z_{n-2} \ldots z_1z_0$ mit n > 0 und $z_i \in Z$ für i=0,1,...,n-1 ist definiert als

$$\phi(z) = \sum_{i=0}^{n-1} \beta(z_i)\cdot b^i$$

Ist b = 10, so spricht man von einem Dezimalsystem.

Sei also (b, Z, β) ein Dezimalsystem, d.h., sei (b, Z, β) mit b = 10, Z = {**0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**} und
β: Z → {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ⊂ ☠🗀 mit
β(**0**) = 0, β(**1**) = 1, β(**2**) = 2, β(**3**) = 3, β(**4**) = 4,
β(**5**) = 5, β(**6**) = 6, β(**7**) = 7, β(**8**) = 8, β(**9**) = 9
ein Stellenwertsystem zur Basis 10.

Der Wert ϕ(z) der Zahldarstellung z = **37** ist daher
$\phi(\mathbf{37}) = \beta(\mathbf{3})\cdot 10^1 + \beta(\mathbf{7})\cdot 10^0$
$= 3\cdot 10^1 + 7\cdot 10^0 = 30 + 7 = 37.$

Der Wert ϕ(z) der Zahldarstellung z = **27188** lautet
$\phi(\mathbf{27188}) = \beta(\mathbf{2})\cdot 10^4 + \beta(\mathbf{7})\cdot 10^3 + \beta(\mathbf{1})\cdot 10^2 + \beta(\mathbf{8})\cdot 10^1 + \beta(\mathbf{8})\cdot 10^0$
$= 2\cdot 10^4 + 7\cdot 10^3 + 1\cdot 10^2 + 8\cdot 10^1 + 8\cdot 10^0 = 27188.$

Ist b = 2, so spricht man von einem Dual- oder Binärsystem.

Sei (b, Z, β) ein Dualsystem, d.h., sei (b, Z, β) mit b = 2,
$Z = \{\mathbf{0}, \mathbf{1}\}$ und $\beta: Z \rightarrow \{0, 1\} \subset \mathbb{N}_0$ mit $\beta(\mathbf{0}) = 0$, $\beta(\mathbf{1}) = 1$.
ein Stellenwertsystem zur Basis 2.

Der Wert $\phi(z)$ der Zahldarstellung $z = \mathbf{101}$ ist daher
$\phi(\mathbf{101}) = \beta(\mathbf{1})\cdot 2^2 + \beta(\mathbf{0})\cdot 2^1 + \beta(\mathbf{1})\cdot 2^0$
$= 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 4 + 1 = 5.$

Der Wert $\phi(z)$ der Zahldarstellung $z = \mathbf{100101}$ lautet
$\phi(\mathbf{100101})$
$= \beta(\mathbf{1})\cdot 2^5 + \beta(\mathbf{0})\cdot 2^4 + \beta(\mathbf{0})\cdot 2^3 + \beta(\mathbf{1})\cdot 2^2 + \beta(\mathbf{0})\cdot 2^1 + \beta(\mathbf{1})\cdot 2^0$
$= 32 + 4 + 1 = 37.$

Ist b = 8, so spricht man von einem Oktalsystem.

Sei (b, Z, β) ein Oktalsystem, d.h., sei (b, Z, β)
mit b = 8, $Z = \{\mathbf{0}, \mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}, \mathbf{5}, \mathbf{6}, \mathbf{7}\}$
und $\beta: Z \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7\} \subset \mathbb{N}_0$ mit
$\beta(\mathbf{0}) = 0$, $\beta(\mathbf{1}) = 1$, $\beta(\mathbf{2}) = 2$, $\beta(\mathbf{3}) = 3$,
$\beta(\mathbf{4}) = 4$, $\beta(\mathbf{5}) = 5$, $\beta(\mathbf{6}) = 6$, $\beta(\mathbf{7}) = 7$
ein Stellenwertsystem zur Basis 8.

Der Wert $\phi(z)$ der Zahldarstellung $z = \mathbf{45}$ ist daher
$\phi(\mathbf{45}) = \beta(\mathbf{4})\cdot 8^1 + \beta(\mathbf{5})\cdot 8^0 = 4\cdot 8 + 5 = 37.$

Der Wert $\phi(z)$ der Zahldarstellung $z = \mathbf{65064}$ lautet
$\phi(\mathbf{65064}) = \beta(\mathbf{6})\cdot 8^4 + \beta(\mathbf{5})\cdot 8^3 + \beta(\mathbf{0})\cdot 8^2 + \beta(\mathbf{6})\cdot 8^1 + \beta(\mathbf{4})\cdot 8^0$
$= 6\cdot 4096 + 5\cdot 512 + 6\cdot 8 + 4 = 27188.$

Ist b = 16, so spricht man von einem Hexadezimalsystem.

Sei also (b, Z, β) ein Hexadezimalsystem, d.h., sei (b, Z, β)
mit b = 16, Z = {**0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **A**, **B**, **C**, **D**, **E**, **F**}
$\beta: Z \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\} \subset \mathbb{N}_0$
mit
β(**0**) = 0, β(**1**) = 1, β(**2**) = 2, β(**3**) = 3, β(**4**) = 4, β(**5**) = 5,
β(**6**) = 6, β(**7**) = 7, β(**8**) = 8, β(**9**) = 9, β(**A**) = 10, β(**B**) = 11,
β(**C**) = 12, β(**D**) = 13, β(**E**) = 14, β(**F**) = 15
ein Stellenwertsystem zur Basis 16.

Der Wert ϕ(z) der Zahldarstellung z = **25** ist daher
$\phi(\mathbf{25}) = \beta(\mathbf{2}) \cdot 16^1 + \beta(\mathbf{5}) \cdot 16^0 = 2 \cdot 16 + 5 = 37.$

Der Wert ϕ(z) der Zahldarstellung z = **6A34** lautet
$\phi(\mathbf{6A34}) = \beta(\mathbf{6}) \cdot 16^3 + \beta(\mathbf{A}) \cdot 16^2 + \beta(\mathbf{3}) \cdot 16^1 + \beta(\mathbf{4}) \cdot 16^0$
$= 6 \cdot 4096 + 10 \cdot 256 + 3 \cdot 16 + 4 = 27188.$

Hinweis 1: In der Regel benutzt man keine gesonderten Zeichen für Z, sondern nimmt die Ziffern von 0 bis b-1 (ab 10 fährt man mit den Buchstaben A, B, ... fort, siehe Hexadezimaldarstellung).

Hinweis 2: Als Basis verwendet man vor allem 10, 2 und 16. Kann es zu Verwechselungen kommen, welche Basis gemeint ist, so schließt man die Zahldarstellung in runde Klammern ein und setzt die Basis b tiefgestellt dahinter:

$(2101)_3$ = "2101 zur Basis 3" = "2·27 + 1·9 + 1" = $(64)_{10}$

$(2101)_4$ = "2101 zur Basis 4" = "2·64 + 1·16 + 1" = $(145)_{10}$

$(2101)_5$ = "2101 zur Basis 5" = "2·125 + 1·25 + 1" = $(276)_{10}$

Hinweis 3: Zu erwähnen sind noch die Darstellungen nach dem "chinesischen Restklassensatz". Wenn m paarweise teilerfremde natürliche Zahlen $q_1, q_2, ..., q_m$ mit $p = q_1 \cdot q_2 \cdot ... \cdot q_m$ gegeben sind, dann lässt sich jede Zahl a mit $0 \le a < p$ *eindeutig* durch das m-Tupel $(r_1, r_2, ..., r_m)$ darstellen, wobei r_i der Rest von a bei der Division durch q_i ist (für i=1,2,...,m). Da man mit den Resten wie mit Zahlen (zyklisch) rechnen kann, könnte dies für umfangreiche compterinterne Berechnungen zu großer Zeitersparnis führen:

Man überträgt die Eingaben in die m-Tupel, rechnet parallel auf allen Komponenten unabhängig voneinander viele Operationen in relativ kurzer Zeit durch und ermittelt am Ende aus den m-Tupeln die zugehörigen Ergebnisse.

Beispiel: m = 3, q_1 = **7**, q_2 = **11**, q_3 = **13**, p = 1001. Die Zahl 400 wird dann durch

(400 mod 7, 400 mod 11, 400 mod 13) = (1,4,10)

repräsentiert; analog werden 10 und 40 durch (3,10,10) bzw. (5,7,1) dargestellt.

Dann ist **10+40** die durch ((3+5) mod 7, (10+7) mod 11, (10+1) mod 13) = (1,6,11) dargestellte Zahl (nämlich 50) und **10·40** die durch ((3·5) mod 7, (10·7) mod 11, (10·1) mod 13) = (1,4,10) dargestellte Zahl (nämlich 400, s.o.) und **40-10** die durch ((5-3) mod 7, (7-10) mod 11, (1-10) mod 13) = (2,8,4) dargestellte Zahl (nämlich 30).

Der schnelleren Ausführung gewisser Operationen stehen auch Nachteile gegenüber, vor allem ist die natürliche Anordnung "<" der Zahlen nicht mehr erkennbar.

Die Darstellung der ganzen Zahlen ($\mathbb{Z}$).

1.1.2.9: *Möglichkeit 1:* Darstellung durch Betrag und Vorzeichen, d.h., wie bei den natürlichen Zahlen zuzüglich eines Vorzeichens "+" oder "-". Auf diese Weise lassen sich mit n Binärstellen die Zahlen von -2^{n-1} bis 2^{n-1} darstellen, wobei die Zahl Null zwei Darstellungen "+0" und "-0" erhält.

Möglichkeit 2: Im Falle der Binärdarstellung (Basis = 2) kann man die erste Stelle der Zahl zur Bildung der negativen Zahlen verwenden, indem man 2^{n-1} abzieht, sofern die erste Stelle eine "1" ist. Der Vorteil dieser zunächst eigenartig wirkenden Darstellung ist, dass die Addition und die Subtraktion mit dem gleichen Algorithmus durchgeführt werden können, während Möglichkeit 1 zwei Verfahren und Anpassungsoperationen benötigt, und dass die Null nur eine Darstellung besitzt.

Möglichkeit 2, Fortsetzung: Der Wert $\phi(z)$ einer n-stelligen binären Zahldarstellung $z = z_{n-1}z_{n-2} \ldots z_1z_0$ mit $n > 0$ und $z_i \in \{0,1\}$ für i=0,1,...,n-1 ist also definiert als

$$\phi(z) = -\beta(z_{n-1})\cdot 2^{n-1} + \sum_{i=0}^{n-2} \beta(z_i)\cdot 2^i$$

1.1.2.10: Diese Darstellung heißt Zwei-Komplementdarstellung und wird oft für die Darstellung von Zahlen in den Registern und Speicherzellen von Computern verwendet.

Hierdurch werden die Zahlen von -2^{n-1} bis $2^{n-1}-1$ beschrieben. Ob eine Zahl negativ ist, erkennt man eindeutig an der ersten Stelle.

Beispiele zur Zwei-Komplementdarstellung:

n = 8, darstellbar sind die Zahlen von -128 bis +127.

0 1 1 1 1 1 1 1 ist die Zahl 127,

1 0 0 0 0 0 0 0 ist die Zahl -128,

1 1 1 1 1 1 1 1 ist die Zahl -1,

0 1 0 1 0 1 0 1 ist die Zahl 85,

1 0 1 0 1 0 1 0 ist die Zahl -86,

1 0 1 0 1 0 1 1 ist die Zahl -85.

Zu einer Zahl x erhält man -x, indem man jede Binärstelle der Darstellung von x komplementiert (also 0 durch 1 und 1 durch 0 ersetzt) und anschließend eine 1 addiert.

1.1.2.11: *Möglichkeit 3:* Verwende eine negative Zahl als Basis in einem Stellenwertsystem gemäß 1.1.2.8. Hierdurch lassen sich alle ganzen Zahlen eindeutig darstellen, sofern b negativ und abs(b) ≥ 2 ist.

Beispiel: Stellenwertsystem zur Basis -2. Seien wie bisher $Z = \{\mathtt{0}, \mathtt{1}\}$ und $\beta(\mathtt{0}) = 0$, $\beta(\mathtt{1}) = 1$. Die ersten Zahldarstellungen:

z	ϕ(z)	z	ϕ(z)	z	ϕ(z)
0	0	110	2	1100	-4
1	1	111	3	1101	-3
10	-2	1000	-8	1110	-6
11	-1	1001	-7	1111	-5
100	4	1010	-10	10000	16
101	5	1011	-9	10001	17

In einem Stellenwertsystem mit negativer Basis gelten ungewöhnliche Regeln; zum Beispiel gilt im System zur Basis -2:

1 + **1** = **110**, **1** + **11** = **0**, **10** + **11** = **1101**,

11 + **11** = **10**, **111** + **111** = **11010**,

100 - **111** = **1** usw.

Wir arbeiten im Dezimalsystem, wobei der Übergang zum Dualsystem keine großen Schwierigkeiten bereitet, da die Operationen +, -, * und div recht ähnlich bleiben. Geht man dagegen zu einer negativen Basis über, so kann man ebenfalls Algorithmen zur Durchführung dieser Operationen angeben, diese sind aber recht ungewohnt. Doch sie könnten sich als günstiger für künftige Computer herausstellen!? Daher muss man möglichst viele Darstellungen für elementare Daten-kennen, untersuchen, erproben usw.