UNIVERSITÄT DORTMUND

Wintersemester 2006/07

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

(alias **Einführung in die Programmierung**)

(Vorlesung)

Prof. Dr. Günter Rudolph

Fachbereich Informatik

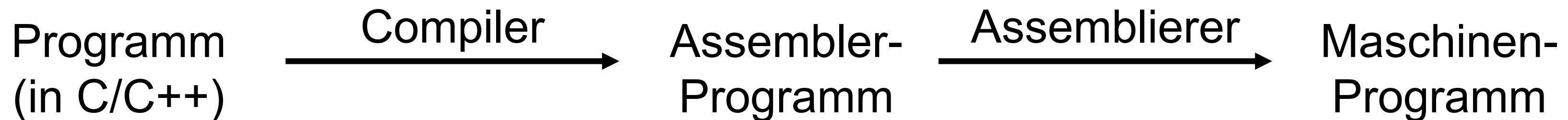
Lehrstuhl für Algorithm Engineering

Inhalt

- Einfache Datentypen
- Zahldarstellungen im Rechner
- Bezeichner
- Datendefinition, Zuweisung, Initialisierung

Realisierung eines Programms

- Problemanalyse
- Spezifikation
- Algorithmenentwurf
- Formulierung eines Programms



- Ausführung erfolgt mit Hilfe des Laufzeitsystems

Notwendig für Programmierung:

- Ausschnitte der realen Welt müssen im Rechner abgebildet werden können!
- Dazu gehören etwa **Daten** in vielerlei Form!
- Bestimmte Formen dieser Daten haben gemeinsame, **typische** Eigenschaften!
- Solche werden zusammengefasst zu so genannten **Datentypen**.

Unterscheidung

- **Einfache Datentypen**

  sind elementar bzw. nicht auf andere Typen zurückführbar.

  Beispiel: positive ganze Zahlen

- **Zusammengesetze Datentypen**

  entstehen baukastenartig durch Zusammensetzen
  von einfachen Datentypen.

  Beispiel: ein Paar aus zwei positiven ganzen Zahlen

Wie werden Zahlen im Rechner dargestellt?

- Bit ∈ { 0, 1 }
- 8 Bit = 1 Byte
- Speicher im Rechner = lineare Folge von Bytes bzw. Bits
- Duales Zahlensystem:
  - n Bits: $(b_{n-1}\ b_{n-2} \ldots b_2\ b_1\ b_0)$ mit $b_k \in \{ 0, 1 \}$
  - 2^n mögliche Kombinationen (= verschiedene Zahlen)
  - Umwandlung in Dezimalzahl:

$$\sum_{k=0}^{n-1} b_k\, 2^k$$

Einfache Datentypen

- **Ganzzahlen ohne Vorzeichen (`unsigned`)**

Bit	Byte	Wertevorrat	Name in C/C++
8	1	0 … 255	**`unsigned char`**
16	2	0 … 65 535	**`unsigned short int`**
32	4	0 ... 4 294 967 295	**`unsigned int`**
32	4	0 ... 4 294 967 295	**`unsigned long int`**

ACHTUNG: Wertebereiche rechnerabhängig! Hier: PC mit Pentium IV.

Negative Zahlen?

- Gleicher Vorrat an verschiedenen Zahlen!

⇒ Vorrat muss anders aufgeteilt werden!

Naiver Ansatz:

- Man verwendet n-1 Bit zur vorzeichenlosen Zahldarstellung
  - ⇒ Das ergibt Zahlen im Bereich 0 … 2^{n-1}-1, also 0 bis 127 für n=8
- Bit n repräsentiert das Vorzeichen: 0 = positiv, 1 = negativ
  - ⇒ Bei n = 8 ergibt das Zahlen im Bereich -127 bis 127
  - ⇒ Probleme:
    - Die Null zählt doppelt: +0 und -0
    - Eine mögliche Zahldarstellung wird verschenkt!

Negative Zahlen?

- Gleicher Vorrat an verschiedenen Zahlen!

⇒ Vorrat muss anders aufgeteilt werden!

`unsigned char`

0 255

-128 0 127

`signed char`

Bitrepräsentation von negativen Zahlen:

- Man muss nur das Stellengewicht des höchstwertigen Bits negativ machen!

Bit	7	6	5	4	3	2	1	0
`unsigned`	128	64	32	16	8	4	2	1
`signed`	-128	64	32	16	8	4	2	1

- Beispiel: 10101001_2 = -128 + 32 + 8 + 1 = -87
- Mit Bit 0 – 6 sind Zahlen zwischen 0 und 127 darstellbar.
  Falls Bit7 = 0 ⇒ 0 bis 127
  Falls Bit7 = 1 ⇒ -128 bis -1

Bitrepräsentation von Ganzzahlen mit Vorzeichen: (n = 8)

7	6	5	4	3	2	1	0	`unsigned`	`signed`
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	1	0	2	2
			…	…				…	…
0	1	1	1	1	1	1	1	127	127
1	0	0	0	0	0	0	0	128	-128
1	0	0	0	0	0	0	1	129	-127
1	0	0	0	0	0	1	0	130	-126
			…	…				…	…
1	1	1	1	1	1	1	1	255	-1

Einfache Datentypen

- **Ganzzahlen mit Vorzeichen**

Bit	Byte	Wertevorrat	Name in C/C++
8	1	-128 … 127	`char`
16	2	-32768 … 32767	`short int`
32	4	-2147483648 ... 2147483647	`int`
32	4	-2147483648 ... 2147483647	`long int`

ACHTUNG: Wertebereiche rechnerabhängig! Hier: PC mit Pentium IV.

Zwischenfragen:

- Wie werden Daten im Programm angelegt bzw. abgelegt?
- Wie kann ich sie wieder finden und abrufen bzw. verändern?

⇒ Rechner muss angewiesen werden Speicherplatz für Daten zu reservieren.

⇒ Das geschieht formal im Programm durch eine **Datendefinition**:

Angabe von **Datentyp** und **Bezeichner**.

Beispiele:

```
char a;
short b;
unsigned long c;
```

Adresse	Daten	Name
11100110	00001001	a
11100101	10001100	
11100100	01101001	b
11100011	10011101	
11100010	11110011	
11100001	10101000	
11100000	00110001	c

Datendefinition (DD)

```
unsigned int Postleitzahl;
```

Was geschieht?

1. DD reserviert Speicher
2. DD legt Wertevorrat fest
3. DD ermöglicht eindeutige Interpretation des Bitmusters
4. DD legt zulässige **Operatoren** fest

Was geschieht nicht?

DD weist keinen Wert zu!

⇒ Zufällige Bitmuster im Speicher! ⇒ Häufige Fehlerquelle!

Zuweisung

- Beispiel: **`Postleitzahl = 44221;`**
- Vor einer Zuweisung muss eine Datendefinition stattgefunden haben!
- Was geschieht?
  - ⇒ Die Zahl wird gemäß Datentyp interpretiert & in ein Bitmuster kodiert.
  - ⇒ Das Bitmuster wird an diejenige Stelle im Speicher geschrieben, die durch den Bezeichner symbolisiert wird.

Initialisierung

- Beispiel: **`unsigned int Postleitzahl = 44221;`**
- Datendefinition mit anschließender Zuweisung

Bezeichner

Bauplan:

- Es dürfen nur Buchstaben `a` bis `z`, `A` bis `Z`, Ziffern `0` bis `9` und der Unterstrich _ vorkommen.
- Das erste Zeichen muss ein Buchstabe oder ein Unterstrich sein.
- Prinzipiell keine Längenbeschränkung.
- **Schlüsselwörter** dürfen nicht verwendet werden.

```
Winkel
EinkomSteuer
Einkom_Steuer
einkom_Steuer
_OK
x3
_x3_und_x4_
_99
```

Schlüsselwörter

… sind reservierte Wörter der jeweiligen Programmiersprache!

`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typeof`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

Schlüsselwörter der Programmiersprache C

Schlüsselwörter

… sind reservierte Wörter der jeweiligen Programmiersprache!

asm	export	private	true
bool	false	protected	try
const_cast	friend	public	typeid
catch	inline	static_cast	typename
class	mutable	template	using
delete	namespace	reinterpret_cast	virtual
dynamic_cast	new	this	
explicit	operator	throw	

Zusätzliche Schlüsselwörter der Programmiersprache C++

Ganzzahlen: Binäre Operatoren

- Addition → Operator: `+`
- Subtraktion → Operator: `-`
- Multiplikation → Operator: `*`
- Ganzzahldivision → Operator: `/`
- Modulo → Operator: `%`

Beispiele:

```
A + b;
3 * x3 - 8 / Faktor;
wert % 12;
```

Ganzzahlen: Modulo-Operator %

- liefert den Rest der Ganzzahldivision
- aus Alltagsleben bekannt, aber selten unter diesem Namen

Beispiel: Digitaluhr

- Wertevorrat: 0:00 bis 23:59
- Stundenanzeige springt nach 23 auf 0
- Minutenanzeige springt nach 59 auf 0
- C/C++:

```
unsigned int stunde, laufendeStunde = 37;
stunde = laufendeStunde % 24;
```

Ganzzahlen: Häufige Fehlerquellen …

- Zahlenüberlauf

```
short m = 400, n = 100, p = 25, k;
k = m * n / p;
```

⇒ Resultat: `k = -1021;` ☹

Warum?

- 400 * 100 ergibt 40000 ⇒ zu groß für Datentyp **`short`** (< 32768)
- 40000 = $1001\ 1100\ 0100\ 0000_2$
- Interpretation als Datentyp short: -32768 + 7232 = -25536
- Schließlich: -25536 / 25 = -1021

Ganzzahlen: Häufige Fehlerquellen …

- Zahlenüberlauf: Addition

```
short a = 32600, b = 200,
c = a + b;
```

⇒ Resultat: `c = -32736;` ☹

- Zahlenüberlauf: Subtraktion

```
unsigned short m = 100, n = 101, k;
k = m - n;
```

⇒ Resultat: `k = 65535;` ☹

Ganzzahlen: Häufige Fehlerquellen …

- Ganzzahldivision ist reihenfolgeabhängig!

Beispiel:

```
20 * 12  /  3          20  /  3  *  12
└──┬───┘               └───┬───┘
  240    /  3              6     *   12
  └─────┬────┘             └─────┬─────┘
       80                        72
```

Merken!

- Wird Zahlenbereich bei Ganzzahlen über- oder unterschritten (auch bei Zwischenergebnissen),
  dann entstehen unvorhersehbare, falsche Ergebnisse
  ohne Fehlermeldung!
- Es liegt im **Verantwortungsbereich des Programmierers**,
  die geeigneten Datentypen auszuwählen (Problemanalyse!).
- Die Verwendung von „größeren“ Datentypen verschiebt das Problem nur auf größere Wertebereiche: es wird i.A. dadurch **nicht gelöst**!
  Es müssen ggf. Vorkehrungen getroffen werden:
  z. B. Konsistenzprüfungen.

Reelle Zahlen

- In C/C++ gibt es zwei Datentypen für reelle Zahlen:

Bit	Byte	Wertebereich	Name in C/C++	Stellen
32	4	$\pm 3.4 * 10^{-38} \ldots \pm 3.4 * 10^{+38}$	`float`	7
64	8	$\pm 1.7 * 10^{-308} \ldots \pm 1.7 * 10^{+308}$	`double`	15

Stellen = signifikante Stellen

Reelle Zahlen

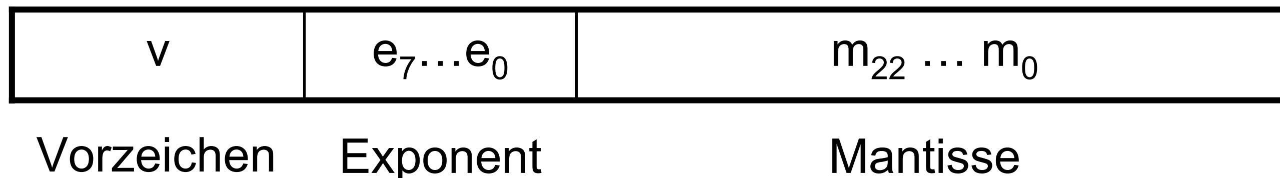
- **`Float`** vs. **`Long`**:

  beide 4 Byte, aber riesiger Unterschied im Wertebereich!

  Wie geht das denn?

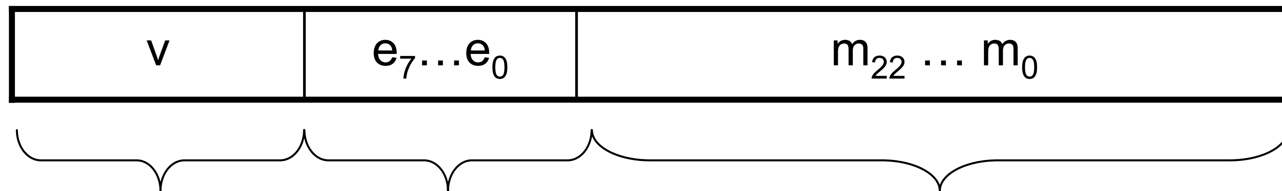
  ⇒ Durch Verlust an Genauigkeit im niederwertigen Bereich der Zahl!

- Repräsentation ist standardisiert: IEEE-Standard P754 (1985)
- Beispiel: **`float`** (32 bit)



Reelle Zahlen

- Repräsentation ist standardisiert: IEEE-Standard P754 (1985)
- Beispiel: **`float`** (32 bit)



	`signed char`	
$0 \Rightarrow +1$	**`-128`**	normiert: $1 \le m \le 2$,
$1 \Rightarrow -1$	⋮ **`+127`**	wobei virtuelles Bit $m_{23} = 1$

Reelle Zahlen

```
float pi1    = 3.141592;
double pi2   = 3.14169265358979;
```

7 signifikante Stellen

15 signifikante Stellen

korrekte

Weitere gültige Schreibweisen:

12345.678 Festkommazahl (*fixed format*)

1.23456e5 Fließkommazahl (*floating point*)

.345

+34.21e-91

Achtung:

Dezimaldarstellung

immer mit Punkt,

niemals mit Komma!

Einfache Datentypen

- **Zeichen**
  - Ein Zeichen wird in einem Byte gespeichert (**`char`**)
  - Zuordnung: Zeichen ↔ Zahl (Code)
  - ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange), 7-Bit-Code

<table>
<tr><td>0</td><td>NUL</td><td>SOH</td><td>STX</td><td>ETX</td><td>EOT</td><td>ENQ</td><td>ACK</td><td>BEL</td><td>BS</td><td>TAB</td><td>LF</td><td>VT</td><td>FF</td><td>CR</td><td>SO</td><td>SI</td></tr>
<tr><td>16</td><td>DLE</td><td>DC1</td><td>DC2</td><td>DC3</td><td>DC4</td><td>NAK</td><td>SYN</td><td>ETB</td><td>CAN</td><td>EM</td><td>SUB</td><td>ESC</td><td>FS</td><td>GS</td><td>RS</td><td>US</td></tr>
<tr><td>32</td><td>SP</td><td>!</td><td>“</td><td>#</td><td>$</td><td>%</td><td>&</td><td>‘</td><td>(</td><td>)</td><td>*</td><td>+</td><td>,</td><td>-</td><td>.</td><td>/</td></tr>
<tr><td>48</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>:</td><td>;</td><td><</td><td>=</td><td>></td><td>?</td></tr>
<tr><td>64</td><td>@</td><td>A</td><td>B</td><td>C</td><td>D</td><td>E</td><td>F</td><td>G</td><td>H</td><td>I</td><td>J</td><td>K</td><td>L</td><td>M</td><td>N</td><td>O</td></tr>
<tr><td>80</td><td>P</td><td>Q</td><td>R</td><td>S</td><td>T</td><td>U</td><td>V</td><td>W</td><td>X</td><td>Y</td><td>Z</td><td>[</td><td>\</td><td>]</td><td>^</td><td>_</td></tr>
<tr><td>96</td><td>`</td><td>a</td><td>b</td><td>c</td><td>d</td><td>e</td><td>f</td><td>g</td><td>h</td><td>i</td><td>j</td><td>k</td><td>l</td><td>m</td><td>n</td><td>o</td></tr>
<tr><td>112</td><td>p</td><td>q</td><td>r</td><td>s</td><td>t</td><td>u</td><td>v</td><td>w</td><td>x</td><td>y</td><td>z</td><td>{</td><td>|</td><td>}</td><td>~</td><td>DEL</td></tr>
</table>

Steuer-zeichen (rows 0 and 16)

Einige wichtige nicht druckbare Steuerzeichen:

horizontal tabulation → TAB; line feed → LF; carriage return → CR

null → NUL; space → SP; delete → DEL

NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
SP	!	“	#	$	%	&	‘	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Zeichen

- Zeichen werden gemäß ihrem Code als Zahl gespeichert

⇒ deshalb kann man mit Zeichen rechnen:

```
char c = '7';

int zahl = c - '0';
```

Code von `'7'` ist 55
Code von `'0'` ist 48

Resultat:
`zahl` = 7

- … und man kann Zeichen vergleichen:

```
'a' < 'b'
```

ist wahr, weil 97 < 98

- Erst bei der Ausgabe wird Datentyp **`char`** wieder als Zeichen interpretiert.

Zeichen

- Datendefinition: **`char Zeichen;`**
- Zuweisung: **`Zeichen = 'x';`**
- Darstellbare Zeichen:
  - Buchstaben: **`'a'`** bis **`'z'`** und **`'A'`** bis **`'Z'`**
  - Ziffern: **`'0'`** bis **`'9'`**
  - Satzzeichen: z.B. **`'!'`** oder **`':'`**
  - Sonderzeichen: z.B. **`'@'`** oder **`'>'`** oder **`'}'`** oder Leerzeichen
  - Steuerzeichen mit Fluchtsymbol (Umschalter): \

`\a`	alarm (BEL)	**`\"`**	Anführungsstriche
`\b`	backspace	**`\'`**	Hochkomma
`\t`	horizontal tabulator (TAB)	**`\?`**	Fragezeichen
`\n`	new line	**`\\`**	backslash

Zeichenketten (Strings)

Datendefinition etc. kommt später!

- Aneinanderreihung von Zeichen
- Gekennzeichnet durch doppelte Hochkommata: "
- Beispiele:
  - `"Dies ist eine Zeichenkette!"`

    ```
    Dies ist eine Zeichenkette!
    ```

  - `"Das ist jetzt\nneu."`

    ```
    Das ist jetzt
    neu.
    ```

  - `"\"The C++ Programming Language\"\n\tby B. Stroustrup"`

    ```
    "The C++ Programming Language"
            by B. Stroustrup
    ```