Modellierung und Programmierung

Dr. Martin Riplinger

24.10.2012

Literatur: Internet

Skripte

- Erik Wallacher: Vorlesungsskript Modellierung/Programmierung
- Gerald Kempfer: Progammieren in C - Vorlesungsbegleitendes Skript
  `http://public.beuth-hochschule.de/~kempfer/skript_c/c.pdf`
- Wikibooks: C-Programmierung
  `http://upload.wikimedia.org/wikibooks/de/8/8d/CProgrammierung.pdf`
- Springerlink: `http://www.springerlink.com`
  - Ralf Kirsch, Uwe Schmitt (Programmieren in C, 2007)
  - Manfred Dausmann, Ulrich Bröckl, Joachim Goll (C als erste Programmiersprache, 2008)
  - Jörg Birmelin, Christian Hupfer (Elementare Numerik für Techniker, 2008)

Literatur: Hardcopy

Semesterapparat

- Modellierung/Programmierung
- Programmierung C/C++
- Fachschaft: Vorlesungsmitschriften

Bücher

- Ralf Kirsch, Uwe Schmitt: Programmieren in C. Eine mathematikorientierte Einführung, Springer 2007 (zurzeit vergriffen, 2. Auflage erscheint 2013)
- Helmut Erlenkötter: C Programmieren von Anfang an, Rowohlt 1999
- Brian Kernighan, Dennis Ritchie: Programmieren in C, Hanser 1990
- Manfred Dausmann, Ulrich Bröckl, Joachim Goll: C als erste Programmiersprache, Teubner 2008

Vom Quellcode zum ausführbaren Programm

Schritt 1: Code schreiben

- Quellcode ist eine Textdatei mit der Endung **.c**
- Ungeeignet: Textverarbeitungsprogramme wie *MS Word* oder *OpenOffice Writer*
- Geeignet: *MS Notepad (Editor), Emacs, gedit, nedit, KDevelop, Eclipse* ...

Schritt 2: Code compilieren

- Kommandozeile starten und in das Verzeichnis wechseln, das den Code enthält
- Befehl `gcc Quellcode.c -o Programm` in eine Kommandozeile eingeben
- Der Compiler gcc übersetzt den Quellcode in Maschinencode, den der Computer versteht.

Schritt 3: Programm ausführen

- Das compilierte Programm wird mit dem Befehl `./Programm` aufgerufen.
- Ohne das vorangestellte `./` liefert das System die Fehlermeldung `bash: Programm: Kommando nicht gefunden.`

Zahlendarstellung: Natürliche Zahlen

$$241012 = 2 \cdot 10^5 + 4 \cdot 10^4 + 1 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 2 \cdot 10^0 = \sum_{j=0}^{N} c_j B^j$$

mit $B = 10$ ($\rightarrow$ Dezimaldarstellung)
$N = 5$
$\mathbf{c} = (c_5, c_4, c_3, c_2, c_1, c_0) = (2, 4, 1, 0, 1, 2)$, $c_j \in \{0, \ldots, B-1\}$

$$24 = 1 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 0 \cdot 2 + 0 \cdot 1$$
$$= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = \sum_{j=0}^{N} c_j B^j$$

mit $B = 2$ ($\rightarrow$ Binärdarstellung)
$N = 4$
$\mathbf{c} = (c_4, c_3, c_2, c_1, c_0) = (1, 1, 0, 0, 0)|_2$, $c_j \in \{0, 1\}$

Zahlendarstellung: Ganze Zahlen

Binärsystem ($B = 2$): c_N wird als Vorzeichen interpretiert.

Beispiel: $B = 2$, $N = 7$

$$-53 = (-1)^1 \cdot (0 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0)$$
$$\rightarrow \mathbf{c} = (1, 0, 1, 1, 0, 1, 0, 1)$$

- Frühe Computer verwendeten diese Darstellung
- Nachteil: 2 Versionen der Null, Ganzzahlarithmetik ineffizient
- Heute üblich: „Zweierkomplement“
  - $\rightarrow$ Eindeutige Darstellung der Null
  - $\rightarrow$ Effizientere Arithmetik, da keine Fallunterscheidung nach Vorzeichen nötig
  - $\rightarrow$ Näheres unter `en.wikipedia.org/wiki/Two's_complement,de.wikipedia.org/wiki/Zweierkomplement`

Zahlendarstellung: Gleitkommazahlen

$$\begin{aligned} -314.15 &= (-1)^1 \cdot 3.1415 \cdot 10^2 \\ &= (-1)^1 \cdot \left(3 \cdot 10^0 + 1 \cdot 10^{-1} + 4 \cdot 10^{-2} + 1 \cdot 10^{-3} + 5 \cdot 10^{-4}\right) \cdot 10^2 \end{aligned}$$

$$\begin{aligned} 0.00027182818 &= (-1)^0 \cdot 2.7182818 \cdot 10^{-4} \\ &= (-1)^0 \left(2 \cdot 10^0 + 7 \cdot 10^{-1} + \ldots + 8 \cdot 10^{-7}\right) \cdot 10^{-4} \\ &= (-1)^0 \left(\sum_{j=0}^{7} m_j 10^{-j}\right) \cdot 10^{-4} \\ &= (-1)^s \left(\sum_{j=0}^{L-1} m_j B^{-j}\right) \cdot B^E \end{aligned}$$

mit	$s = 0$	Vorzeichen
	$B = 10$	Basis
	$E = -4$	Exponent
	$\mathbf{m} = (m_0, m_1, \ldots, m_{L-1}) = (2,7,1,8,2,8,1,8)$	Mantisse ($m_0 \neq 0$)
	$L = 8$	Mantissenlänge, $L = \lvert\mathbf{m}\rvert$

$\boxed{\mathbf{x} = (s, \mathbf{m}, E)_B}$ eindeutig!

Zahlendarstellung: Gleitkommazahlen

Im Computer: Basis $B = 2$ (Auch für Darstellung des Exponenten als Ganzzahl!)
$m_0 \neq 0 \Rightarrow m_0 = 1$ (wird nicht gespeichert!)

Darstellungsformel für Gleitkommazahlen im Computer

$$x = (-1)^s \cdot \left(1 + \sum_{j=1}^{L} m_j 2^{-j}\right) \cdot 2^{E + E_{\min}}$$

mit $s, m_j \in \{0,1\}$, $E \in \{0,1,\ldots,2^k - 1\}$, $E_{\min} = -2^{k-1} + 1$ (min. Exponent).

Reservierte Werte:

- $\mathbf{m} = (0,\ldots,0)$, $E = 0$ $\longrightarrow$ `+0.0` oder `-0.0`
- $\mathbf{m} \neq (0,\ldots,0)$, $E = 0$ $\longrightarrow$ „Subnormale“ Zahl
- $\mathbf{m} = (1,\ldots,1)$, $E = 2^k - 1$ $\longrightarrow$ `+inf` oder $-$`inf` (unendlich)
- $\mathbf{m} \neq (1,\ldots,1)$, $E = 2^k - 1$ $\longrightarrow$ `NaN` („Not a number“)

„Echte“ Gleitkommazahlen für Exponenten $1 \leq E \leq 2^k - 2$

$\Rightarrow\ -2^{k-1} + 2 \leq E + E_{\min} \leq 2^{k-1} - 1$ (zwischen -126 und 127 für $k = 8$)

Zahlendarstellung: Rechnen mit Gleitkommazahlen

Addition ($L = |\mathbf{m}| = 7,\ B = 10$)

$fl(123456.7) \oplus fl(101.7654)$

$= 1.234567 \cdot 10^5 \oplus 1.017654 \cdot 10^2$ Floating Point Darstellung

$= 1.234567 \cdot 10^5 \oplus 0.001017654 \cdot 10^5$ Exponentenausgleich

$= fl\,(1.234567 + 0.001017654) \cdot 10^5$

$= fl\,(1.235584654) \cdot 10^5$

$= 1.235585 \cdot 10^5$ Floating Point Darstellung

Subtraktion ($L = |\mathbf{m}| = 7,\ B = 10$) $a = 123457.1467,\ b = 123456.659$

$$fl(a) \ominus fl(b) = 1.234571 \cdot 10^5 \ominus 1.234567 \cdot 10^5$$
$$= 0.000004 \cdot 10^5$$
$$= 4.000000 \cdot 10^{-1}$$

Relativer Fehler: $\dfrac{|\,|fl(a) \ominus fl(b)| - |a-b|\,|}{|a-b|} = \dfrac{|0.4 - 0.4877|}{0.4877} \approx 0.18 = 18\%$!!

Zahlendarstellung: Rechnen mit Gleitkommazahlen

Assoziativgesetz: $(a+b)+c \stackrel{?}{=} a+(b+c)$

Es seien: $L = 4,\ B = 10$

$a = 1.0 \quad = 1.000 \cdot 10^0$

$b = 0.0003 = 3.000 \cdot 10^{-4}$

$c = 0.0004 = 4.000 \cdot 10^{-4}$

$$\left\{fl(a) \oplus fl(b)\right\} \oplus fl(c) = 1.000 \cdot 10^0 + 3.000 \cdot 10^{-4} = 1.000 \cdot 10^0$$
$$fl(a) \oplus \left\{fl(b) \oplus fl(c)\right\} = 1.000 \cdot 10^0 + 7.000 \cdot 10^{-4} = 1.001 \cdot 10^0$$

$\longrightarrow$ Reihenfolge der Operationen ist relevant!

Datentypen

Definition

Ein Datentyp ist festgelegt durch einen Wertebereich und die darauf anwendbaren Operationen.

Datentyp	"Wertebereich"
`char`	Zeichen (bspw. Buchstaben und Ziffern)
`int`	ganze Zahlen
`float`	Gleitkommazahlen mit einfacher Genauigkeit
`double`	Gleitkommazahlen mit doppelter Genauigkeit

Typmodifizierer	erlaubt für
`short`	`int`
`long`	`int`, `double`
`long long`	`int`
`unsigned`	`int`, `char`, `short int`, `long int`, `long long int`

Datentypen

Datentyp	**Größe**	**kleinster Wert**	**größter Wert**
`char`	1 Byte (=8 Bit)	$-2^7 = -128$	$2^7 - 1 = 127$
`unsigned char`	1 Byte	0	$2^8 - 1 = 255$
`int`	4 Byte	-2^{31}	$2^{31} - 1$
`unsigned` (`int`)	4 Byte	0	$2^{32} - 1$
`long` (`int`)	4 Byte*	wie `int`	wie `int`
`unsigned long`	4 Byte*	wie `unsigned`	wie `unsigned`
`long long`	8 Byte	-2^{63}	$2^{63} - 1$
`unsigned long long`	8 Byte	0	$2^{64} - 1$

* Gilt für 32-Bit-Systeme (Linux/Windows/Mac) sowie 64-Bit-Windows. Auf 64-Bit-Linux- oder Mac-Systemen hat `long` eine Größe von 8 Byte = 64 Bit.

Datentyp	**Größe**	**betragsmäßig kleinster Wert**	**betragsmäßig größter Wert**
`float`	4 Byte	$\approx 1.18 \cdot 10^{-38}$	$\approx 3.40 \cdot 10^{38}$
`double`	8 Byte	$\approx 2.23 \cdot 10^{-308}$	$\approx 1.80 \cdot 10^{308}$
`long double`	10 Byte**	$\approx 3.36 \cdot 10^{-4932}$	$\approx 1.19 \cdot 10^{4932}$

** Nicht eindeutig festgelegt. Der ISO-Standard verlangt lediglich, dass `long double` **mindestens** die gleiche Präzision aufweist wie `double`. Die meisten Compiler interpretieren `long double` als 80-Bit-Gleitkommazahl.

Variablendeklaration

Definition

Eine Variablendeklaration besteht aus der Angabe eines Datentyps sowie einer Liste von Variablennamen.

```
Datentyp Variablenname1, Variablenname2,...,VariablennameN;
```

- vor der ersten Verwendung
- zu Beginn eines Anweisungsblocks
- Unterscheidung zwischen Groß- und Kleinschreibung
- keine Sonderzeichen, wie z.B. #, ß, %
- keine Ziffern zu Beginn
- max. Variablennamenlänge: 31 Zeichen

Positiv-Beispiele:

- `int i, j, k;`
- `float u, w = -3.14, x, y = 1.0, z;`
- `unsigned int N = 5; double s;`

Variablendeklaration

Negativ-Beispiele:

`int dummy#;`	`Fehler: verirrtes »#« im Programm`
`int dummy%;`	`Fehler: expected »=«, »,«, »;« ...`
`int i, double dummy;`	`Fehler: expected identifier or »(« before »double«`
`int i; double 2dummy;`	`Fehler: ungültiger Suffix »dummy« an Ganzzahlkonstante`
`unsigned double x;`	`Fehler: both »unsigned« and »double« in declaration specifiers`

„Feste Variablen“: `const`

```
const Datentyp Variablenname = Wert;
```

Auf die entsprechende Variable kann nur noch lesend zugegriffen werden. Bsp.:

```
const int k = 1;
  ⋮
k = 2;          → Fehler: Zuweisung der schreibgeschützten Variable »k«
```

Operatoren

Definition

- Anweisung/Manipulation/Rechnung mit festen Regeln
- wirkt auf einen oder mehrere Operanden
  - unärer Operator: 1 Operand
  - binärer Operator: 2 Operanden
- Stellung des Operators:
  - Präfixform: Operator steht vor Operanden
  - Suffixform: Operator steht hinter Operanden
  - Infixform: Operator steht zwischen Operanden

Zuweisungsoperator '='

```
int a;
float x = 3.14, y, z;

a = -4;
z = y = x;
 // aequivalent: z = (y = x);
```

- binär, Infixstellung
- rechtsassoziativ ($\leftarrow$)

Operatoren: Arithmetische Operatoren

Arithmetische Opertoren (+ , - , * , / , %)

Name	Verwendung	Operandentyp	Resultat
Minus (unär)	`-Op1`	`int`, `float`	Vorzeichenwechsel
Plus	`Op1 + Op2`	`int`, `float`	Summe
Minus (binär)	`Op1 - Op2`	`int`, `float`	Differenz
Multiplikation	`Op1 * Op2`	`int`, `float`	Produkt
Division	`Op1 / Op2`	`int` / `float`	ganzzahlige Division / Quotient
Modulo	`Op1 % Op2`	`int`	Rest bei ganzzahliger Division

Regeln

- Sind die Operanden vom gleichen Datentyp, so auch das Ergebnis
- Sind die Operanden von verschiedenen Typen, so ist das Resultat vom „genaueren“Datentyp: 'int' + 'double' = 'double'
- Arithmetische Operatoren sind linksassoziativ ($\rightarrow$) , d.h. $a+b+c=(a+b)+c$
- Priorität bei binären Operatoren: „Punkt vor Strich“

Operatoren: Arithmetische Zuweisung und Inkrement

Arithmetische Zuweisungsoperatoren

Operation	Bezeichnung	Äquivalent zu
`Op1 += Op2;`	Additionszuweisung	`Op1 = Op1 + Op2;`
`Op1 -= Op2;`	Subtraktionszuweisung	`Op1 = Op1 - Op2;`
`Op1 *= Op2;`	Multiplikationszuweisung	`Op1 = Op1 * Op2;`
`Op1 /= Op2;`	Divisionszuweisung	`Op1 = Op1 / Op2;`
`Op1 %= Op2;`	Modulozuweisung	`Op1 = Op1 % Op2;`

Inkrement- und Dekrementoperatoren ++ und --

Für eine Variable `i` vom Typ `int` sind äquivalent:

`i = i+1;` $\iff$ `i += 1;` $\iff$ `i++;`

`i = i-1;` $\iff$ `i -= 1;` $\iff$ `i--;`

Unterscheidung Postfix- und Präfixnotation

`sum += i++;` $\iff$
```
sum = sum + i;
i = i + 1;
```

`sum += ++i;` $\iff$
```
i = i + 1;
sum = sum + i;
```

Operatoren: Arithmetische Zuweisung und Inkrement

```
int a = 3, b, c;
b = ++a * 3; // a = 4, b = 12
c = a++ * 3; // c = 12, a = 5
```

ist äquivalent zu

```
int a = 3, b, c;
a++;   // oder ++a;
b = a * 3;
c = a * 3;
++a;   // oder a++;
```

Publikumsfrage: Seien `a` und `b` vom Typ `int`. Was ist zu `b -= (a-1);` äquivalent?

a) `-b  = a - 1;`

b) `b -= (-a);`

c) `b  = -(a--);`

d) `b -= (--a);`

Operatoren: Division und Modulorechnung

Mathematik (Zahlentheorie):
Seien $a,\ 0 \neq b \in \mathbb{Z}$. Dann existieren eindeutig bestimmte Zahlen $q, r \in \mathbb{Z}$ mit

$$a = q \cdot b + r, \qquad 0 \leq r < |b| \ .$$

Computer Science:
Seien $a,\ 0 \neq b \in \mathbb{Z}$. Dann existieren (eindeutig bestimmte) Zahlen $q, r \in \mathbb{Z}$ mit

$$a = q \cdot b + r, \qquad -|b| < r < |b| \ .$$

Es gilt:

```
q = a / b;
r = a % b;
```

$\rightarrow$ ganzzahlige Division
$\rightarrow$ a modulo b

Beispiele:

- $a = 45$, $b = 7$:
  $45 = 6 \cdot 7 + 3 \quad \rightsquigarrow \quad q = 6\,, \quad r = \ \ 3$
  $45 = 7 \cdot 7 - 4 \quad \rightsquigarrow \quad q = 7\,, \quad r = -4$
- $a = -27$, $b = 5$:
  $-27 = (-6) \cdot 5 + 3 \quad \rightsquigarrow \quad q = -6\,, \quad r = \ \ 3$
  $-27 = (-5) \cdot 5 - 2 \quad \rightsquigarrow \quad q = -5\,, \quad r = -2$

Operatoren: Implizite Typumwandlung und Casts

Unterscheide!

```
int
a = -3;
float c;
c = a / 2;
```

$\rightsquigarrow$ `c = -1.0`

```
int
a = -3;
float c;
c = a / 2.0;
```

$\rightsquigarrow$ `c = -1.5`

Explizite Typumwandlung durch Casts

```
(Datentyp) Term;
```

Beispiel:

```
int   a = 3, b = 2;
float c = (float) a / b;
```

$\rightsquigarrow$ `c = 1.5`

Merke: Casts besitzen höhere Priorität als binäre arithmetische Operatoren

- **Äquivalent:** `float c = ((float) a) / b;`
- **Nicht äquivalent:** `float c = (float) (a / b);`

Operatoren: Vergleichende und logische Operatoren

Vergleichsoperatoren

- Überprüft werden Wahrheitswerte von Aussagen wie etwa $x > 0$, $j \leq N$, $a \leq f(x) \leq b$ usw.

Notation in C	math. Notation
`a < b`	$a < b$
`a > b`	$a > b$
`a <= b`	$a \leq b$
`a >= b`	$a \geq b$
`a == b`	$a = b$
`a != b`	$a \neq b$

- Häufig verwendet in „*wenn*, . . . *dann* . . . "-Konstruktionen
- in C: Wert `0` (auch `0.0`) $\rightsquigarrow$ "falsch" (`false`)
  Werte ungleich `0` (etwa `1`, `-2.5`) $\rightsquigarrow$ "wahr" (`true`)
- Unterscheide "`a = b`" (Zuweisung) und "`a == b`" (Vergleich)
  Merke: Der Wert einer Zuweisung entspricht dem zugewiesenen Wert.
- Vorsicht beim Test auf Gleichheit bei floats (s. arithm. Operatoren)
- Vergleichsoperatoren sind linksassoziativ ($\longrightarrow$)

Operatoren: Vergleichende und logische Operatoren

Logische Operatoren: `&&` , `||` , `!`

- Ausdrücke (arithmetische, vergleichende, zuweisende) werden als Aussagen miteinander verknüpft
- Die Werte solcher Verknüpfungen sind immer `1` (wahr) oder `0` (falsch)

Notation in C	math. Notation	Bedeutung
`A && B`	$A \wedge B$	Konjunktion (und)
`A || B`	$A \vee B$	Disjunktion (oder)
`!A`	$\neg A$	Negation

- Das "oder" ist einschließend zu verstehen und nicht als "entweder oder"
- Logische Operatoren sind linksassoziativ ($\longrightarrow$)
- **Verknüpfungstafeln:**

&&	0	1
0	0	0
1	0	1

||	0	1
0	0	1
1	1	1

	!
0	1
1	0

Operatoren: Vergleichende und logische Operatoren

Beispiel:

```
int A = -4, C = 1;
double B = -0.5;
```

Ausdruck	Wert
A	-4
!A	0
A - B	-3.5
!(A - B)	0
!!(A - B)	1
!A - B	0.5
!A -!B	0

Ausdruck	Wert
A < B < C	0
A < B <= C	1
A < (B<C)	1
A<B && B<C	1
C>A && B	1
A==2 || B>C	0
A=2 || B>C	1

Ausgabe auf der Kommandozeile: `printf`

- dient zur Ausgabe von Text und numerischen Werten auf dem Bildschirm
- erfordert Präprozessordirektive: `#include <stdio.h>`
- Syntax: `printf(Formatstring, Parameterliste);`
- `Formatstring` = Text und Platzhalter in Anführungszeichen

Beispiel:

```
int a = -5;
float b = 3.1415926535;

printf("a hat den Wert %d, b hat den Wert %f.\n", a, b);
```

Ausgabe:

```
a hat den Wert -5, b hat den Wert 3.141593.
```

Zeichenkonstanten (vgl. Erlenkötter, Kap. 8.5)

`\n` neue Zeile (new line)

`\t` horizontaler Tabulator

Ein- und Ausgabe: Platzhalter

```
%[flags][weite][.genauigkeit][modifizierer]typ
```

- `typ`

ganzzahlig	`d` / `i`	3218
unsigned integer	`u`	3218
Gleitkomma	`f`	3218.000000
wissenschaftl.	`e`	3.218000e+03

- `modifizierer` `l`, z.B. `%lf` bei Verwendung von `double`
  `L`, z.B. `%Lf` bei Verwendung von `long double`
- `genauigkeit` Anzahl der Nachkommastellen
- `weite` Mindestanzahl an Zeichen („`.`“ mitgezählt)
- `flags` links- oder rechtsbündig, Vorzeichen, führende Nullen

Beispiel: `"%7.3f"` ist Platzhalter für eine Gleitkommazahl mit 3 Nachkommastellen und einer Feldbreite von mindestens 7 Zeichen.

Hilfe zu `printf`: Befehl `man 3 printf` in die Kommandozeile eingeben

Weitere Beispiele: Übung