Einführung in die Programmierung

Bachelor of Science

Prof. Dr. Rethmann

Fachbereich Elektrotechnik und Informatik
Hochschule Niederrhein

WS 2009/10

Inhalt

Literatur

Informatik allgemein

- Peter Rechenberg: Was ist Informatik?
  Carl Hanser Verlag
- H.P. Gumm, M. Sommer: Einführung in die Informatik
  Oldenbourg Verlag

Die Programmiersprache C

- Jürgen Wolf: C von A bis Z.
  Galileo Computing.
- B.W. Kernighan, D.M. Ritchie: Programmieren in C
  Carl Hanser Verlag
- K. Zeiner: Programmieren lernen mit C
  Carl Hanser Verlag

Übersicht

Inhalt

Ein erstes kleines Programm

Hello, World!

```
#include <stdio.h>

void main(void) {
    printf("Hello,\nWorld!\n");
}
```

Erklärungen:

- Mittels `#include <stdio.h>` wird eine Bibliothek bereitgestellt, die Funktionen zur Ein- und Ausgabe enthält.
- Der Start eines Programms besteht im Ausführen der Funktion `main`.
- Alle Anweisungen werden mit einem Semikolon beendet.

Ein erstes kleines Programm

Erklärungen: (Fortsetzung)

- Die Funktion `printf()` gibt eine Zeichenkette auf dem Bildschirm aus.
  Solche Standardfunktionen sind übersetzte Funktionen, die zur C-Implementierung gehören.
- eine Zeichenkette ist durch doppelte Anführungsstriche am Anfang und Ende gekennzeichnet, z.B. `"James Bond"`
- Anweisungsfolgen werden mit geschweiften Klammern `{` und `}` zusammengefasst, der geklammerte Block gilt als eine Anweisung.
- das Zeichen `\n` bedeutet *new line*, es bewirkt also einen Zeilenvorschub

Ein zweites kleines Programm

Variablen:

```
#include <stdio.h>
void main(void) {
    int i = 5;
    printf("i = %d\n", i);
    i = i + 2;
    printf("i = %d\n", i);
}
```

Erklärungen:

- alle Variablen in C haben einen Typ, im Beispiel definieren wir `i` vom Typ ganze Zahl
- Variablen können in Ausdrücken wie `i = i + 2` verwendet werden: erhöhe den Wert von `i` um zwei und weise diesen neuen Wert der Variablen `i` zu

Ein zweites kleines Programm

Hinweis:

- Variablen in C nehmen zu unterschiedlichen Zeiten unterschiedliche Werte an

```
int i = 1;
while (i < 10) {
    printf("sqr(%d) = %d\n", i, i * i);
    i = i + 1;
}
```

- Variablen in der Mathematik sind Platzhalter für feste Werte

$$\begin{aligned} 3x + 4y &= 20 \\ 2x + 17y &= 42 \end{aligned}$$

also $x = 4$, $y = 2$

Ein drittes kleines Programm

Schleifen:

```
#include <stdio.h>
void main(void) {
    int i = 10;
    while (i <= 80) {
        printf("%3d | %5.2f\n", i, i * 1.35962);
        i = i + 10;
    }
}
```

Erklärungen:

- solange der Wert von `i` kleiner oder gleich 80 ist, wird der Rumpf der Schleife ausgeführt
- sobald der Wert von `i` größer als 80 ist, wird die Abarbeitung des Schleifenrumpfs abgebrochen (evtl. schon zu Beginn)

Ein drittes kleines Programm

Anwendung z.B. bei Tabellen:

```
#include <stdio.h>
void main(void) {
    printf(" 10 |   7.35\n");
    printf(" 20 |  14.71\n");
    printf(" 30 |  22.06\n");
    printf(" 40 |  29.42\n");
    printf(" 50 |  36.77\n");
    printf(" 60 |  44.13\n");
    printf(" 70 |  51.48\n");
    printf(" 80 |  58.84\n");
}
```

Schleifen machen Programme kürzer und besser wartbar bzw. erweiterbar

Ein viertes kleines Programm

Zählschleifen:

```
#include <stdio.h>
void main(void) {
    int i;
    for (i = 1; i < 10; i += 1) {
        printf("i = %d\n", i);
    }
}
```

Erklärungen:

- der erste Ausdruck ist der *Initialisierungsausdruck*, der vor Beginn der Schleife einmal ausgeführt wird
- solange der zweite Ausdruck erfüllt ist, wird der Schleifenrumpf durchlaufen
- nach jedem Schleifendurchlauf wird der dritte Ausdruck bewertet → Schleifenvariablen ändern

Ein fünftes kleines Programm

Leerzeilen:

```
#include <stdio.h>

void main(void) {
    int i = 5;

    while (i < 10) {
        printf("i = %d\n", i);
        i = i + 2;
    }
}
```

Erklärungen: Leerzeilen haben keine syntaktische Bedeutung, aber sie erhöhen die Lesbarkeit des Programms

- vor Funktionen
- nach Deklaration von Variablen

Ein sechstes kleines Programm

```
#include <stdio.h>

void main(void) {
    int i, anf, end;

    printf("Anfangswert? ");
    scanf("%d", &anf);
    printf("Endwert? ");
    scanf("%d", &end);

    i = anf;
    while (i < end) {
        printf("i = %d\n", i);
        i = i + 1;
    }
}
```


Ein sechstes kleines Programm

Erklärung: die Funktion `scanf()` liest Werte von der Tastatur ein

- dazu muss zum einen der erwartete Datentyp angegeben werden

`%d`	`int`
`%f`	`float`
`%c`	`char`

- zum anderen das Ziel, also die Variable, in der der eingelesene Wert gespeichert werden soll
  dabei ist unbedingt auf das Ampersand `&` zu achten

Ein siebtes kleines Programm

Verzweigungen:

```
#include <stdio.h>

void main(void) {
    int i;

    printf("Wert? ");
    scanf("%d", &i);

    if (i % 2 == 0)
        printf("%d ist gerade\n", i);
    else printf("%d ist ungerade\n", i);
}
```


Ein siebtes kleines Programm

Erklärungen:

- Mittels Auswahlanweisungen kann der Ablauf eines Programms abhängig von Bedingungen geändert werden.
- Der Modulo-Operator `%` bestimmt den ganzzahligen Rest bei einer Division.
- Im Beispiel wird der Ausdruck `i % 2 == 0` bewertet. Falls er wahr ist, wird die darauf folgende Anweisung ausgeführt, ansonsten die Anweisung im `else`-Zweig
- Der `else`-Zweig kann entfallen

Übersicht

Inhalt

- erste, einfache C-Programme
- *Studieren: wie geht das?*

ECTS-Punkte

*E*uropean *C*redit *T*ransfer *S*ystem

- ein ECTS-Punkt steht für 30 Stunden Lernaufwand
- für jedes Semester gibt es 30 ECTS-Punkte

⇒ 900 Stunden Lernaufwand pro Semester

Ihr Studium ist ein Vollzeitjob!

denn der typische Vorlesungszeitraum beträgt nur 15 Wochen

⇒ 60 Stunden Lernaufwand pro Woche

Grundlagen des Lernens

Wissen kann nicht übertragen werden, es muss im Gehirn eines jeden Lernenden neu geschaffen werden

folgende Faktoren bestimmen den Lernerfolg:

- eine positive Einstellung zum Lernen
- Vertrauenswürdigkeit des Lehrenden und des Lernortes
- der gegenwärtige emotionale Zustand des Lernenden
- die spezielle Motiviertheit für einen bestimmten Stoff

Lernen findet nur statt, wenn das Gehirn des Lernenden einen Gewinn bzw. Sinn im Lernen allgemein und im Erwerb des speziellen Lerninhalts sieht!

Gehirngerechtes Lehren und Lernen

- *Überblick vor Einzelinformation*
  Damit das Gehirn nach schon vorhandenen Speicherplätzen suchen kann.
- *Transparenz der Lehr- und Lernziele*
  Mit Verständnis für den Sinn des Lernens wird man lernbereiter.
- *Interesse wecken*
  Neugierde ist die beste Voraussetzung, um Neues aufzunehmen und zu behalten.
- *Wiederholen*
  Wenn Nervenschaltkreise öfter betätigt werden, werden sie stabiler.

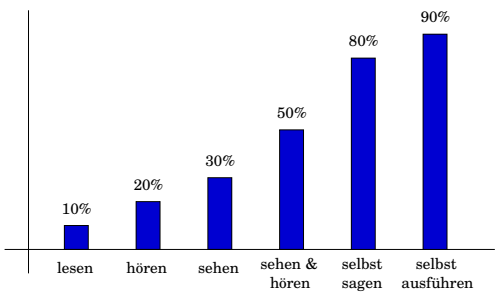
Gehirngerechtes Lehren und Lernen

- *Mehrere Sinne ansprechen*
  Informationen sollten nicht nur über Auge und Ohr, sondern auch über das Selber-Machen ins Gehirn gelangen.
  → *Übungen und Praktika*
- *Auf die Gefühle achten*
  Angst und Stress behindern das Gedächtnis, positive Gefühle unterstützen die Speicherung und den Abruf von Informationen.
- *Rückmelden*
  Eine möglichst baldige Rückmeldung, ob das Richtige gelernt wurde, ermöglicht noch Korrekturen im Prozess der Speicherung. Lob, Verstärkung und Bekräftigung sind wichtig.
  → *Lerngruppen*

Gehirngerechtes Lehren und Lernen

- *Pausen einlegen*
  Zeit und Ruhe sind zur Konsolidierung/Festigung des Stoffes notwendig. Während einer Pause sollten keine ähnlichen Informationen aufgenommen werden.
  → *nicht Zeitung lesen, sondern Fenster putzen*
- *In richtiger Reihenfolge lehren und lernen*
  Ein roter Faden bei den Lernschritten bewirkt im Gehirn eine sinnvolle Vernetzung des Stoffes mit dem dazu passenden alten Bereich.
- *Vernetzen*
  Das Gehirn arbeitet assoziativ und vernetzt. Lernen sollte deshalb in Zusammenhängen, fächerübergreifend und projektorientiert stattfinden.

Aktives Lernen



Je aktiver Sie lernen, um so mehr behalten Sie!

zur Vorlesung

Aktuelle Informationen, Sprechzeiten, Folien unter

`http://lionel.kr.hsnr.de/~rethmann/index.html`

Anmerkungen, Korrekturen, Verbesserungsvorschläge sind immer willkommen! Sprechen Sie mich an oder schicken Sie mir eine E-Mail.

E-Mail: `jochen.rethmann@hsnr.de`
Büro: B329

Stellen Sie Fragen! Nur so kann ich beurteilen, ob Sie etwas verstanden haben oder noch im Trüben fischen.

Was können/müssen Sie tun?

Wie schaffen Sie es, den Stoff der Vorlesung zu verstehen und zu behalten?

Idee: Sie sagen laut: „Hey, Gehirn, egal wie langweilig dieses Buch auch ist, und egal wie gering der Ausschlag auf meiner emotionalen Richterskala gerade ist, ich will wirklich, dass du diesen Kram behälst.“

der langsame, ermüdende Weg: Wiederholung

- Sie hämmern sich die gleiche Sache immer wieder ein
- irgendwann sagt Ihr Gehirn: „Er hat zwar nicht das *Gefühl*, dass das wichtig ist, aber er sieht es sich *so oft an*, dann muss es wohl wichtig sein.“

Was können/müssen Sie tun?

der schnelle, effektive Weg: Gehirnaktivität erhöhen

- *intensiv nachdenken:*
  Falls Sie nicht aktiv Ihre Neuronen strapazieren, passiert in Ihrem Gehirn nicht viel.
  Stellen Sie sich Fragen: Was soll das? Was hat er da denn wieder gemacht? Warum macht er das nicht anders?
- *visualisieren Sie den Text:*
  Ihr Gehirn ist auf visuelle Eindrücke eingestellt: Ein Bild sagt mehr als 1024 Worte!
- *nehmen Sie aktiv an Übungen und Praktika teil:*
  Ihr Gehirn lernt und behält besser, wenn Sie Dinge tun, als wenn Sie nur darüber lesen.

Was können/müssen Sie tun?

- *trinken Sie viel Wasser:*
  Ihr Gehirn arbeitet am besten in einem schönen Flüssigkeitsbad. Austrocknung beeinträchtigt die kognitive Funktion.
  Austrocknung beginnt schon vor dem ersten Durst!
- *reden Sie darüber:*
  Sprechen aktiviert einen anderen Teil des Gehirns.
  Noch besser: Versuchen Sie es jemandem zu erklären. Sie lernen dann schneller und haben vielleicht Ideen, auf die Sie beim bloßen Lesen nie gekommen wären.
  → *Lerngruppen!*
- *entwerfen Sie etwas:*
  Wenden Sie das Gelernte an. Tun Sie irgendetwas, um neben den Übungen und Praktika weitere Erfahrungen zu sammeln.

Übersicht

Zahlendarstellung im Rechner

- *Motivation*
- Codierung
- Stellenwertsysteme
- Zahlenumwandlung
- Addition und Multiplikation
- Ganze Zahlen
- Gleitpunktzahlen

Motivation

Im digitalen Computer werden nur Nullen und Einsen gespeichert.

- Um Buchstaben oder Zahlen in der uns gewohnten Form anzuzeigen, müssen diese Symbole codiert werden.
- Die uns bekannten Rechenverfahren wie Addition und Multiplikation müssen auf Dualzahlen durchgeführt werden.

Aber: Warum ist die interne Speicherung der Daten auch bei der Programmentwicklung interessant?

Motivation

Welche Ausgabe erzeugt das folgende Programm?

```
#include <stdio.h>

int main(void) {
    float x = 1.0f;

    printf("   X   | X * X\n");
    printf("-------+-------\n");
    while (x != 2.0f) {
        printf("%6.2f | %6.2f\n", x, x * x);
        x += 0.1f;
    }
    return 0;
}
```


Motivation

```
   X   | X * X
-------+-------
     ...
  1.40 |   1.96
  1.50 |   2.25
  1.60 |   2.56
  1.70 |   2.89
  1.80 |   3.24
  1.90 |   3.61
  2.00 |   4.00
  2.10 |   4.41
  2.20 |   4.84
  2.30 |   5.29
  2.40 |   5.76
     ...
```


Motivation

Welche Ausgabe erzeugt das folgende Programm?

```
#include <stdio.h>

int main(void) {
    int x, fakultaet = 1;

    printf("  X |     X!\n");
    printf("----+-------------\n");
    for (x = 1; x <= 20; x++) {
        fakultaet *= x;
        printf("%3d | %12d\n", x, fakultaet);
    }
    return 0;
}
```

Motivation

```
  X |     X!
----+-------------
   ...
  9 |       362880
 10 |      3628800
 11 |     39916800
 12 |    479001600
 13 |   1932053504
 14 |   1278945280
 15 |   2004310016
 16 |   2004189184
 17 |   -288522240
 18 |   -898433024
 19 |    109641728
 20 |  -2102132736
```


Motivation

Welche Ausgabe erzeugt das folgende Programm?

```c
#include <stdio.h>

int main(void) {
    int x, i = 16777211;
    float f = 16777211.0;

    printf("      i     |       f\n");
    printf("------------+-------------\n");
    for (x = 0; x < 20; x++) {
        printf(" %9d | %12.1f\n", i, f);
        i += 1;
        f += 1.0;
    }
    return 0;
}
```


Motivation

```
      i     |       f
------------+-------------
  16777211 |   16777211.0
  16777212 |   16777212.0
  16777213 |   16777213.0
  16777214 |   16777214.0
  16777215 |   16777215.0
  16777216 |   16777216.0
  16777217 |   16777216.0
  16777218 |   16777216.0
  16777219 |   16777216.0
    ....
```

Also: Um zu verstehen, warum diese Programme sich so verhalten, müssen wir uns mit der Zahlendarstellung im Rechner befassen.

Übersicht

Zahlendarstellung im Rechner

- Motivation
- *Codierung*
- Stellenwertsysteme
- Zahlenumwandlung
- Addition und Multiplikation
- Ganze Zahlen
- Gleitpunktzahlen

Allgemeines

Alphabet: In Daten vorkommende Zeichen gehören immer einer bestimmten Zeichenmenge an:

- Zahlen → Ziffern, Dezimalpunkt und Vorzeichen
- Texte → Buchstaben, Ziffern und Satzzeichen

Das klassische Alphabet der indogermanischen Kultur ist

$$\Sigma_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Weitere Alphabete:

Σ_2	=	$\{0, 1\}$	dual, binär
Σ_8	=	$\{0, 1, 2, \ldots, 7\}$	oktal
Σ_{16}	=	$\{0, 1, 2, \ldots, 9, A, B, C, D, E, F\}$	hexadezimal

Codierung

Die Symbole aller denkbaren Alphabete lassen sich durch *Gruppen von Binärzeichen* ausdrücken.

Beispiel: Das deutsche Alphabet der Kleinbuchstaben kann wie folgt dargestellt werden:

00000	a	00100	e	01000	i
00001	b	00101	f	01001	j
00010	c	00110	g	01010	k
00011	d	00111	h	01011	l ...

Mit Gruppen aus n Binärzeichen lassen sich 2^n Symbole codieren.

Codierung

Ein *Code* ist also die Zuordnung einer Menge von Zeichenfolgen zu einer anderen Menge von Zeichenfolgen.

Die Zuordnung (besser Abbildung) erfolgt oft durch eine Tabelle, der *Codetabelle*.

Um Ziffern, Klein- und Großbuchstaben, Umlaute, Satzzeichen sowie einige mathematische Zeichen zu codieren, reichen 8 Binärzeichen aus.

International:

- früher: *ASCII* (American Standard Code for Information Interchange)
- heute: *Unicode* (gebräuchlich ist UTF-8)

ASCII-Tabelle (Auszug)

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
054	44	2C	,	070	56	38	8
055	45	2D	-	071	57	39	9
056	46	2E	.	072	58	3A	:
057	47	2F	/	073	59	3B	;
060	48	30	0	074	60	3C	<
061	49	31	1	075	61	3D	=
062	50	32	2	076	62	3E	>
063	51	33	3	077	63	3F	?
064	52	34	4	100	64	40	@
065	53	35	5	101	65	41	A
066	54	36	6	102	66	42	B
067	55	37	7	103	67	43	C

Codierung

Warum ist diese Art der Codierung für Zahlen ungeeignet? Zahlen sind doch Zeichenketten, die wir im ASCII codieren können.

Codierung

Warum ist diese Art der Codierung für Zahlen ungeeignet? Zahlen sind doch Zeichenketten, die wir im ASCII codieren können.

- *sehr speicherintensiv:* 123 dargestellt in

ASCII-Codierung:	00110001 00110010 00110011	3 Byte
Binärcodierung:	01111011	1 Byte

- *komplexe Arithmetik:*
  - ASCII-Werte können nicht einfach summiert werden

$$\begin{array}{rr} 8 & 00111000 \;\hat{=}\; 8 \\ +\,9 & +\,00111001 \;\hat{=}\; 9 \\ \hline 17 & 01110001 \;\hat{=}\; \mathrm{q} \end{array}$$

  - Wie geht man mit Überträgen um?

$$\begin{array}{rr} & 17 \\ + & 43 \\ \hline & 60 \end{array} \qquad \begin{array}{rrr} & 49 & 55 \\ + & 52 & 51 \\ \hline & 101 & 106 \end{array}$$

Übersicht

Inhalt

b-adische Darstellung natürlicher Zahlen

Satz: Sei $b \in \mathbb{N}$ und $b > 1$. Dann ist jede ganze Zahl z mit $0 \leq z \leq b^n - 1$ und $n \in \mathbb{N}$ eindeutig als Wort $z_{n-1}z_{n-2}\ldots z_0$ der Länge n über Σ_b dargestellt durch:

$$z = z_{n-1} \cdot b^{n-1} + z_{n-2} \cdot b^{n-2} + \ldots + z_1 \cdot b^1 + z_0 \cdot b^0$$

(ohne Beweis)

Vereinbarungen:

- In der Ziffernschreibweise geben wir in der Regel die Basis der Zahlendarstellung explizit an, außer wenn eindeutig klar ist, welche Basis gemeint ist.
- Die Basis selbst wird immer dezimal angegeben.

b-adische Darstellung natürlicher Zahlen

Beispiel: Sei $b = 10$ (Dezimalsystem).
Die eindeutige Darstellung von $z = 4711$ lautet

$$\begin{aligned} z &= 4 \cdot 10^3 + 7 \cdot 10^2 + 1 \cdot 10^1 + 1 \cdot 10^0 \\ &= 4 \cdot 1000 + 7 \cdot 100 + 1 \cdot 10 + 1 \cdot 1 \\ &= 4000 + 700 + 10 + 1 \end{aligned}$$

und in Ziffernschreibweise $(4711)_{10}$.

b-adische Darstellung natürlicher Zahlen

Beispiel: Sei $b = 2$ (Dualsystem).
Die eindeutige Darstellung von $z = 42$ lautet

$$\begin{aligned} z &= 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \\ &= 1 \cdot 32 + 0 \cdot 16 + 1 \cdot 8 + 0 \cdot 4 + 1 \cdot 2 + 0 \cdot 1 \\ &= 1 \cdot 32 + 1 \cdot 8 + 1 \cdot 2 \end{aligned}$$

und in Ziffernschreibweise $(101010)_2$.

b-adische Darstellung natürlicher Zahlen

Beispiel: Sei $b = 8$ (Oktalsystem).
Die eindeutige Darstellung von $z = 93$ lautet

$$\begin{aligned} z &= 1 \cdot 8^2 + 3 \cdot 8^1 + 5 \cdot 8^0 \\ &= 1 \cdot 64 + 3 \cdot 8 + 5 \cdot 1 \\ &= 64 + 24 + 5 \end{aligned}$$

und in Ziffernschreibweise $(135)_8$.

b-adische Darstellung natürlicher Zahlen

Beispiel: Sei $b = 16$ (Hexadezimalsystem).
Die eindeutige Darstellung von $z = 342$ lautet

$$\begin{aligned} z &= 1 \cdot 16^2 + 5 \cdot 16^1 + 6 \cdot 16^0 \\ &= 1 \cdot 256 + 5 \cdot 16 + 6 \cdot 1 \\ &= 256 + 80 + 6 \end{aligned}$$

und in Ziffernschreibweise $(156)_{16}$.

Übersicht

Inhalt

Dezimalsystem → andere Systeme

Beispiel: $(935.421875)_{10} = (3A7.6C)_{16}$.

1. Zahl aufteilen in Vor- und Nachkommateil.
2. Vorkommateil durch fortgesetzte Division umwandeln.

$$\begin{array}{rcrcrcl} 935 & : & 16 & = & 58 & \text{Rest } 7 & \mathrel{\hat{=}} 7 \\ 58 & : & 16 & = & 3 & \text{Rest } 10 & \mathrel{\hat{=}} A \\ 3 & : & 16 & = & 0 & \text{Rest } 3 & \mathrel{\hat{=}} 3 \end{array}$$

Die jeweiligen Divisionsreste ergeben von unten nach oben gelesen den Vorkommateil der gesuchten Zahl in der anderen Basis.

Dezimalsystem → andere Systeme

3. Nachkommateil durch fortgesetzte Multiplikation umwandeln.

$$\begin{array}{rcrcrcrcl} 0.421875 & \cdot & 16 & = & 6 & + & 0.75 & \to & 6 \\ 0.75 & \cdot & 16 & = & 12 & + & 0 & \to & C \end{array}$$

Die jeweiligen ganzen Teile ergeben von oben nach unten gelesen den Nachkommateil der gesuchten Zahl in der anderen Basis.

Übung: zeigen Sie die Korrektheit des Verfahrens

Dezimalsystem → andere Systeme

Beispiel: $(978.127685546875)_{10} = (3D2.20B)_{16}$
Vorkommateil:

$$\begin{array}{rcrcrcl} 978 & : & 16 & = & 61 & \text{Rest } 2 & \mathrel{\hat{=}} 2 \\ 61 & : & 16 & = & 3 & \text{Rest } 13 & \mathrel{\hat{=}} D \\ 3 & : & 16 & = & 0 & \text{Rest } 3 & \mathrel{\hat{=}} 3 \end{array}$$

Nachkommateil:

$$\begin{array}{rcrcrcrcl} 0.127685546875 & \cdot & 16 & = & 2 & + & 0.04296875 & \to & 2 \\ 0.04296875 & \cdot & 16 & = & 0 & + & 0.6875 & \to & 0 \\ 0.6875 & \cdot & 16 & = & 11 & + & 0 & \to & B \end{array}$$

Dezimalsystem → andere Systeme

Beispiel: $(122.1)_{10} = (172.0\overline{6314})_8$
Vorkommateil:

$$\begin{array}{rcrcrcl} 122 & : & 8 & = & 15 & \text{Rest } 2 \\ 15 & : & 8 & = & 1 & \text{Rest } 7 \\ 1 & : & 8 & = & 0 & \text{Rest } 1 \end{array}$$

Nachkommateil:

$$\begin{array}{rcrcrcrcl} 0.1 & \cdot & 8 & = & 0 & + & 0.8 & & \\ 0.8 & \cdot & 8 & = & 6 & + & 0.4 & & \\ 0.4 & \cdot & 8 & = & 3 & + & 0.2 & & \\ 0.2 & \cdot & 8 & = & 1 & + & 0.6 & & \\ 0.6 & \cdot & 8 & = & 4 & + & 0.8 & \ldots & \end{array}$$

Periodische Dualbrüche

Bei der Umwandlung vom Dezimal- ins Dualsystem ergeben sich oft periodische Dualbrüche: $(0.1)_{10} = (0.0\overline{0011})_2$.

Im Rechner:

- da die Länge der Zahlen beschränkt ist, können periodische Dualbrüche nur näherungsweise dargestellt werden

⇒ welche Auswirkungen das haben kann, sieht man am einführenden Beispiel im Abschnitt „Motivation“

- bei n Nachkommastellen ist der durch das Weglassen weiterer Dualstellen entstehende Fehler im Mittel die Hälfte der letzten dargestellten Ziffer $\to 0.5 \cdot 2^{-n}$

beliebige Systeme → Dezimalsystem

Berechnen der b-adischen Darstellung: *Horner-Schema*

Beispiel: $(63D2)_{16} = (25554)_{10}$

$$\begin{aligned} (63D2)_{16} &= 6 \cdot 16^3 + 3 \cdot 16^2 + 13 \cdot 16^1 + 2 \cdot 16^0 \\ &= (6 \cdot 16 + 3) \cdot 16^2 + 13 \cdot 16^1 + 2 \cdot 16^0 \\ &= [(6 \cdot 16 + 3) \cdot 16 + 13] \cdot 16 + 2 \end{aligned}$$

Durch Anwenden des Horner-Algorithmus reduziert sich die Anzahl der durchzuführenden Multiplikationen.

Beispiel: $(1736)_8 = (990)_{10}$

$$\begin{aligned} (1736)_8 &= 1 \cdot 8^3 + 7 \cdot 8^2 + 3 \cdot 8^1 + 6 \cdot 8^0 \\ &= [(1 \cdot 8 + 7) \cdot 8 + 3] \cdot 8 + 6 \end{aligned}$$

Umwandlung artverwandter Systeme

Bei zwei Basen $b, b' \in \mathbb{N}$ mit $b' = b^n$ für ein $n \in \mathbb{N}$ kann die Zahlenumwandlung vereinfacht werden.

Beispiel: $(21121,012)_3 = (247,16)_9$, also $b = 3$ und $b' = 9$

Die Ziffern der Zahl $(21121,012)_3$ werden paarweise zusammengefasst, da $9 = 3^2$.

- dem Vorkommateil evtl. Nullen voran stellen
- an den Nachkommateil ggf. Nullen anfügen

Vorkomma		Nachkomma		
$(02)_3$	$=$ $(2)_9$	$(01)_3$	$=$	$(1)_9$
$(11)_3$	$=$ $(4)_9$	$(20)_3$	$=$	$(6)_9$
$(21)_3$	$=$ $(7)_9$			

Umwandlung artverwandter Systeme

Beispiel: $(32132)_4 = (39E)_{16}$
Die Ziffern der Zahl $(32132)_4$ werden paarweise zusammengefasst, da $16 = 4^2$.

$$\begin{aligned}(03)_4 &= (3)_{16}\\ (21)_4 &= (9)_{16}\\ (32)_4 &= (E)_{16}\end{aligned}$$

Umwandlung artverwandter Systeme

Beispiel: $(2A7)_{16} = (0010\ 1010\ 0111)_2$
Die Ziffern der Zahl $(2A7)_{16}$ werden jeweils als 4-stellige Dualzahl geschrieben, da $16 = 2^4$:

$$\begin{aligned}(2)_{16} &= (0010)_2\\ (A)_{16} &= (1010)_2\\ (7)_{16} &= (0111)_2\end{aligned}$$

Übersicht

Zahlendarstellung im Rechner

- Motivation
- Codierung
- Stellenwertsysteme
- Zahlenumwandlung
- *Addition und Multiplikation*
- Ganze Zahlen
- Gleitpunktzahlen

Addition

Addition einstelliger Dualzahlen:

0	0	1	1	
+ 0	+ 1	+ 0	+ 1	
0	1	1	10	Übertrag

Beispiel: Addition mehrstelliger Zahlen

Dezimal		Dual	
	37		00100101
	+49		+00110001
	86		01010110

Multiplikation

Multiplikation einstelliger Dualzahlen:

$$\begin{aligned}0 * 0 &= 0\\ 1 * 0 &= 0\\ 0 * 1 &= 0\\ 1 * 1 &= 1\end{aligned}$$

Beispiel: Multiplikation mehrstelliger Zahlen

Dezimal	$37 \cdot 21$	Dual	$00100101 \cdot 00010101$
	37		100101
	740		10010100
	777		1001010000
			1100001001

Subtraktion und Division

Beispiel: ggf. auch negative Ergebnisse zulassen

3	3	3	3	3		3
- 0	- 1	- 2	- 3	- 4	...	- 9
3	2	1	0	-1		-6

aber: dann können wir nicht stellenweise vorgehen

23
- 14
?? -1

Subtraktion und Division

Beispiel: ggf. eins leihen von nächst höherer Stelle

3	3	3	3
- 0	- 1	- 2	- 3
3	2	1	0

13	13	13		13
- 4	- 5	- 6	...	- 9
9	8	7		4

Übertrag

aber: wir dürfen nur kleinere von größeren Zahlen subtrahieren

23	23	33
- 9	- 16	- 36
1	1	1
14	7	??7

Subtraktion und Division

Bei der Darstellung mit Vorzeichen und Betrag benötigen wir eine Entscheidungslogik, ob addiert oder subtrahiert werden muss.

vier Fälle sind zu unterscheiden:

Operanden	Operation	Beispiel
$+x, +y$	$x + y$	$5 + 2 = 5 + 2$
$-x, -y$	$-(x + y)$	$-5 - 2 = -(5 + 2)$
$+x, -y$ mit $\lvert x\rvert \geq \lvert y\rvert$	$x - y$	$5 - 2 = 5 - 2$
$-x, +y$ mit $\lvert y\rvert \geq \lvert x\rvert$	$y - x$	$-2 + 5 = 5 - 2$
$+x, -y$ mit $\lvert x\rvert < \lvert y\rvert$	$-(y - x)$	$2 - 5 = -(5 - 2)$
$-x, +y$ mit $\lvert y\rvert < \lvert x\rvert$	$-(x - y)$	$-5 + 2 = -(5 - 2)$

weitere Nachteile dieser Darstellung:

- zwei Nullen: -0 und $+0$
- wir benötigen Addierer und Subtrahierer

Übersicht

Zahlendarstellung im Rechner

- Motivation
- Codierung
- Stellenwertsysteme
- Zahlenumwandlung
- Addition und Multiplikation
- *Ganze Zahlen*
- Gleitpunktzahlen

Komplementdarstellung

Im folgenden ist n immer eine fest gewählte Zahl, d.h. wir arbeiten immer mit Zahlen fester Wortlänge, z.B. 16 Bit.

Sei $x = x_{n-1}x_{n-2}\dots x_0$ eine n-stellige Dualzahl. Sei

$$\overline{x_i} = \begin{cases} 1 & \text{falls } x_i = 0 \\ 0 & \text{falls } x_i = 1 \end{cases}$$

Einer-Komplement:

$$\overline{x}^1 = \begin{cases} x_{n-1}x_{n-2}\dots x_0 & \text{, falls } x \geq 0 \\ \overline{x_{n-1}}\,\overline{x_{n-2}}\dots\overline{x_0} & \text{, sonst} \end{cases}$$

Zweier-Komplement:

$$\overline{x}^2 = \begin{cases} x_{n-1}x_{n-2}\dots x_0 & \text{, falls } x \geq 0 \\ \overline{x_{n-1}}\,\overline{x_{n-2}}\dots\overline{x_0} + 1 \text{ (modulo } 2^n) & \text{, sonst} \end{cases}$$

Komplementdarstellung

Beispiel: 8 Bit-Wortlänge

$$\begin{aligned} x &= -00110100 \\ \overline{x}^1 &= 11001011 \\ \overline{x}^2 &= 11001100 \end{aligned}$$

Beispiel: 16 Bit-Wortlänge

$$\begin{aligned} x &= -0001101101011000 \\ \overline{x}^1 &= 1110010010100111 \\ \overline{x}^2 &= 1110010010101000 \end{aligned}$$

Eine Komplementdarstellung ist immer auf eine beliebige, aber fest vorgegebene Stellenzahl bezogen.

Komplementdarstellung

Für jede b-adische Zahlendarstellung kann das $(b-1)$- und das b-Komplement definiert werden.

Komplementdarstellung: eine negative Zahl $-x$ wird durch die Differenz $N - x$ dargestellt (N = Anzahl darstellbarer Zahlen)

Beispiel: Für $b = 10$ und $n = 3$ gilt $N = 10^3 = 1000$.

-23 entspricht im Zehner-Komplement $N - 23 = 977$

$$\begin{array}{r} 568 \\ -\ 23 \\ \hline 545 \end{array} \qquad \begin{array}{r} 568 \\ +\ 977 \\ \hline 1545 \end{array}$$

Komplementdarstellung

Eine Komplementdarstellung ist immer auf eine beliebige, aber fest vorgegebene Stellenzahl bezogen.

Dualzahl	Einer-Komplement	Zweier-Komplement
000 = 0	000 = 0	000 = 0
001 = 1	001 = 1	001 = 1
010 = 2	010 = 2	010 = 2
011 = 3	011 = 3	011 = 3
100 = 4	100 = -3	100 = -4
101 = 5	101 = -2	101 = -3
110 = 6	110 = -1	110 = -2
111 = 7	111 = -0	111 = -1

Subtraktion im Einer-Komplement

Voraussetzung: x und y zwei n-stellige Dualzahlen

1. anstelle von $x - y$ berechne $\overline{x}^1 + \overline{y}^1$
2. ggf. Übertrag zur niederwertigsten Stelle addieren

Beispiel: Sei $n = 8$. Aus der Rechnung

$$\begin{array}{rrcr} x & 01110111 & \widehat{=} & 119 \\ -y & -00111011 & \widehat{=} & 59 \\ \hline & 00111100 & \widehat{=} & 60 \end{array}$$

wird im Einer-Komplement

$$\begin{array}{rr} \overline{x}^1 & 01110111 \\ +\overline{y}^1 & +11000100 \\ \hline & 100111011 \end{array} \qquad \begin{array}{r} 00111011 \\ +1 \\ \hline 00111100 \end{array}$$

Subtraktion im Einer-Komplement

Beispiel: Aus der Rechnung

$$\begin{array}{rr} 27 & 00011011 \\ -38 & -00100110 \\ \hline -11 & -00001011 \end{array}$$

wird im Einer-Komplement

$$\begin{array}{rcr} 00011011 & \widehat{=} & 27 \\ +11011001 & \widehat{=} & -38 \\ \hline 11110100 & \widehat{=} & -11 \end{array}$$

Subtraktion im Einer-Komplement

Übung: Aus der Rechnung

$$\begin{array}{rr} -17 & -00010001 \\ -38 & -00100110 \\ \hline -55 & -00110111 \end{array}$$

wird im Einer-Komplement ...

Subtraktion im Zweier-Komplement

Voraussetzung: x und y zwei n-stellige Dualzahlen

1. anstelle von $x - y$ berechne $\bar{x}^2 + \bar{y}^2$
2. ignoriere eventuell auftretenden Übertrag

Beispiel: Sei $n = 8$. Aus der Rechnung

$$\begin{array}{rrcr} x & 01110111 & \hat{=} & 119 \\ -y & -00111011 & \hat{=} & 59 \\ \hline & 00111100 & \hat{=} & 60 \end{array}$$

wird im Zweier-Komplement

$$\begin{array}{rr} \bar{x}^2 & 01110111 \\ +\bar{y}^2 & +11000101 \\ \hline & 100111100 \end{array}$$

Subtraktion im Zweier-Komplement

Beispiel: Aus der Rechnung

$$\begin{array}{rr} 27 & 00011011 \\ -38 & -00100110 \\ \hline -11 & -00001011 \end{array}$$

wird im Zweier-Komplement

$$\begin{array}{rrcr} & 00011011 & \hat{=} & 27 \\ +11011010 & & \hat{=} & -38 \\ \hline & 11110101 & \hat{=} & -11 \end{array}$$

Subtraktion im Zweier-Komplement

Übung: Aus der Rechnung

$$\begin{array}{rr} -17 & -00010001 \\ -38 & -00100110 \\ \hline -55 & -00110111 \end{array}$$

wird im Zweier-Komplement ...

Ganze Zahlen im Rechner

Und mit folgendem Programm können wir testen, ob ganze Zahlen im Rechner tatsächlich so abgelegt werden.

```
#include <stdio.h>

void main(void) {
    int i, z;

    printf("Wert? ");
    scanf("%d", &z);

    for (i = 31; i >= 0; i--) {
        if (z & (1 << i))
            printf("1");
        else printf("0");
    }
    printf("\n");
}
```


Übersicht

Zahlendarstellung im Rechner

- Motivation
- Codierung
- Stellenwertsysteme
- Zahlenumwandlung
- Addition und Multiplikation
- Ganze Zahlen
- *Gleitpunktzahlen*

Darstellung von Gleitpunktzahlen

Operationen auf *Festkomma*-Zahlen: Das Komma muss bei allen Operanden an der gleichen Stelle stehen.

Beispiel: Addition der Zahlen 101.01 und 11.101:

$$\begin{array}{rcr} 101 & . & 01 \\ + \quad 11 & . & 101 \\ \hline 1000 & . & 111 \end{array}$$

Also: Zahlen müssen eventuell transformiert werden
⇒ signifikante Stellen gehen verloren

Beispiel: Bei 4 Vor- und 4 Nachkommastellen muss die Zahl 0.000111101 durch 0000.0001 abgerundet dargestellt werden

→ fünf signifikante Stellen gehen verloren

Halblogarithmische Darstellung

Bei der *Gleitpunktdarstellung* wird jede Zahl z dargestellt in der Form:

$$z = \pm m \cdot b^{\pm d}$$

m : *Mantisse*
d : *Exponent*
b : *Basis des Exponenten*

Beispiele:

$$\begin{array}{rcrcr} 3.14159 & = & 0.314159 & \cdot & 10^{1} \\ 0.000021 & = & 0.21 & \cdot & 10^{-4} \\ 12340000 & = & 0.1234 & \cdot & 10^{8} \end{array}$$

Halblogarithmische Darstellung

Die halblogarithmische Darstellung ist nicht eindeutig:

$$\begin{array}{rcrcr} 3.14159 & = & 0.0314159 & \cdot & 10^{2} \\ & = & 0.314159 & \cdot & 10^{1} \\ & = & 31.4159 & \cdot & 10^{-1} \\ & = & 314.159 & \cdot & 10^{-2} \end{array}$$

Mehrdeutigkeiten vermeiden → normalisierte Darstellung

Eine Gleitpunktzahl der Form $\pm m \cdot b^{\pm d}$ heißt *normalisiert*, wenn gilt:

$$\frac{1}{b} \leq |m| < 1$$

Halblogarithmische Darstellung

Beispiele:

$$
\begin{aligned}
(0.000011101)_2 &\rightarrow (0.11101)_2 \cdot 2^{-4} \\
(1001.101)_2 \cdot 2^{10} &\rightarrow (0.1001101)_2 \cdot 2^{14} \\
3.14159 &\rightarrow 0.314159 \cdot 10^{1} \\
47.11 \cdot 10^{2} &\rightarrow 0.4711 \cdot 10^{4} \\
0.0815 \cdot 10^{-3} &\rightarrow 0.815 \cdot 10^{-4}
\end{aligned}
$$

Gleitpunktzahlen

Speichern einer Gleitpunktzahl: aufteilen der 32 Bit wie folgt:

1. *Mantisse:* 23 Bit für den Betrag plus ein Bit für das Vorzeichen der Mantisse (Vorzeichen-/Betragdarstellung)
2. *Exponent:* 8 Bit (Zweier-Komplement Darstellung)
3. erste Stelle der Mantisse ist immer Null und wird in der Rechnerdarstellung ignoriert

analog für 64 Bit-Darstellung ...

Gleitpunktzahlen

Beispiel:

$$
\begin{aligned}
(5031.1875)_{10} &= (1001110100111.0011)_2 \cdot 2^{0} \\
&= (0.10011101001110011)_2 \cdot 2^{13} \\
&= (0.10011101001110011)_2 \cdot 2^{(00001101)_2}
\end{aligned}
$$

Vorzeichen	:	$0 \mathrel{\hat{=}} +$
Mantisse	:	10011101001110011000000
Exponent	:	00001101

Gleitpunktzahlen

Beispiel:

$$
\begin{aligned}
(-0.078125)_{10} &= (-0.000101)_2 \cdot 2^{0} \\
&= (-0.101)_2 \cdot 2^{-3} \\
&= (-0.101)_2 \cdot 2^{(11111101)_2}
\end{aligned}
$$

Vorzeichen	:	$1 \mathrel{\hat{=}} -$
Mantisse	:	10100000000000000000000
Exponent	:	11111101

Gleitpunktzahlen

Null im darstellbaren Bereich nicht enthalten
⇒ abweichende Darstellung: Vorzeichen 0, Mantisse 0, Exponent 0

Bei $b = 2$ ist das erste Bit der Mantisse immer 1:

- erstes Bit der Mantisse nicht speichern (*hidden bit*)
- Verwechselung zwischen $\frac{1}{2}$ und 0 ausschließen

Gleitpunktzahlen haben Einbußen hinsichtlich der Genauigkeit:

- größte Gleitpunktzahl bei 32 Bit: etwa 2^{127}
- größte Zahl in Dualdarstellung bei 32 Bit: $2^{31} - 1$

Was das für die Programmierung heißt, haben wir bereits im einführenden Beispiel im Abschnitt Motivation.

Arithmetik

Seien $x = m_x \cdot 2^{d_x}$ und $y = m_y \cdot 2^{d_y}$.

Multiplikation:
Mantissen multiplizieren, Exponenten addieren

$$x \cdot y = (m_x \cdot m_y) \cdot 2^{d_x + d_y}$$

Division:
Mantissen dividieren, Exponenten subtrahieren

$$x : y = (m_x : m_y) \cdot 2^{d_x - d_y}$$

Arithmetik

Beispiel: $12.18 \cdot 3.7$

$$
\begin{aligned}
0.1218 \cdot 10^{2} \cdot 0.37 \cdot 10^{1} &= 0.045066 \cdot 10^{3} \\
&= 45.066
\end{aligned}
$$

Beispiel: $45.066 : 3.7$

$$
\begin{aligned}
0.45066 \cdot 10^{2} : 0.37 \cdot 10^{1} &= 1.218 \cdot 10^{1} \\
&= 12.18
\end{aligned}
$$

Arithmetik

Seien $x = m_x \cdot 2^{d_x}$ und $y = m_y \cdot 2^{d_y}$.

Addition:

$$x + y = (m_x \cdot 2^{d_x - d_y} + m_y) \cdot 2^{d_y} \quad \text{falls} \quad d_x \leq d_y$$

Subtraktion:

$$x - y = (m_x \cdot 2^{d_x - d_y} - m_y) \cdot 2^{d_y} \quad \text{falls} \quad d_x \leq d_y$$

kleiner Exponent muss großem Exponenten angeglichen werden
⇒ Rundungsfehler durch Denormalisierung

Rundungsfehler

Bei einer Genauigkeit von 4 Stellen seien x, y, z wie folgt gegeben:

$$\begin{aligned} x &= +0.1235 \cdot 10^3 \\ y &= +0.5512 \cdot 10^5 \\ z &= -0.5511 \cdot 10^5 \end{aligned}$$

Dann gilt

$$\begin{aligned} x + y &= +0.1235 \cdot 10^3 + 0.5512 \cdot 10^5 \\ &= +0.0012 \cdot 10^5 + 0.5512 \cdot 10^5 \\ &= +0.5524 \cdot 10^5 \end{aligned}$$

$$\begin{aligned} (x + y) + z &= +0.5524 \cdot 10^5 - 0.5511 \cdot 10^5 \\ &= +0.0013 \cdot 10^5 \\ &= +0.1300 \cdot 10^3 \end{aligned}$$

Rundungsfehler

Andererseits gilt

$$\begin{aligned} y + z &= +0.5512 \cdot 10^5 - 0.5511 \cdot 10^5 \\ &= +0.0001 \cdot 10^5 \\ &= +0.1000 \cdot 10^2 \end{aligned}$$

$$\begin{aligned} x + (y + z) &= +0.1235 \cdot 10^3 + 0.1000 \cdot 10^2 \\ &= +0.1235 \cdot 10^3 + 0.0100 \cdot 10^3 \\ &= +0.1335 \cdot 10^3 \\ &\neq +0.1300 \cdot 10^3 \end{aligned}$$

Das Assoziativ-Gesetz gilt also nicht bei Gleitpunktzahlen!

Ungenauigkeiten bei Gleitpunktzahlen

Ungenauigkeiten im Umgang mit Gleitpunktzahlen

- bei der Umwandlung vom Dezimal- ins Dualsystem
- und bei den arithmetischen Operationen.

In der Regel spielen kleine Abweichungen keine große Rolle. Im Rechner werden aber oft tausende von Rechenoperationen hintereinander ausgeführt: kleine Rundungsfehler können sich addieren und das Resultat völlig unbrauchbar machen!

Exzess-Darstellung

- zur Darstellung des Exponenten bei Gleitpunktzahlen
- zum Wert einer Zahl x wird eine positive Zahl q addiert, so dass das Ergebnis nicht negativ ist
- Exzess q gleich Betrag der größten negativen Zahl

Beispiel: Anzahl Bits gleich 4 $\Rightarrow q = 8$

x	$x+q$	Code	x	$x+q$	Code	x	$x+q$	Code
-8	0	0000	-3	5	0101	2	10	1010
-7	1	0001	-2	6	0110	3	11	1011
-6	2	0010	-1	7	0111	4	12	1100
-5	3	0011	0	8	1000	..	..	..
-4	4	0100	1	9	1001	7	15	1111

IEEE-754: Gleitpunktzahlen im Rechner

- Exponent in Exessdarstellung speichern
- normalisiert wird auf 1.xxxxxx
- die führende Eins wird nicht abgespeichert
- einfache Genauigkeit (32Bit)
  - Exzess: $2^7 - 1 = 127$
  - 1Bit Vorzeichen, 8 Bit Exponent, 23 Bit Mantisse
- doppelte Genauigkeit (64Bit)
  - Exzess: $2^{10} - 1 = 1023$
  - 1Bit Vorzeichen, 11 Bit Exponent, 52 Bit Mantisse

IEEE-754: Gleitpunktzahlen im Rechner

Beispiel: einfache Genauigkeit

$$(-0.078125)_{10} = (-0.000101)_2 \cdot 2^0 = (-1.01)_2 \cdot 2^{-4}$$

Vorzeichen	:	$1 \mathrel{\hat{=}} -$
Exponent	:	01111011 $(\mathrel{\hat{=}} -4 + 127)$
Mantisse	:	01000000000000000000000

Ergänzende Literatur:
Walter Oberschelp und Gottfried Vossen: Rechneraufbau und Rechnerstrukturen, Oldenbourg Verlag