5 Gleitpunktzahlen

Mit der *IEEE*-Gleitpunktarithmetik wurde in ein Standard eingeführt, der heute auf (praktisch) allen Rechnern implementiert ist.

5.1 Definition, Rundung und Rundungsfehler

Gleitpunktzahlen Sei $g \in \mathbb{N},\ g \geq 2$, die Basis des verwendeten Zahlsystems, $t \in \mathbb{N}$ die Mantissenlänge und $m, M \in \mathbb{Z}$ die Schranken für den Exponenten. Die Menge der Gleitpunktzahlen $\mathcal{G}(g, t, m, M)$ umfaßt alle Zahlen der Form

$$z = \ \mathrm{sgn}(z) g^l \sum_{\nu=1}^{t} a_{-\nu} g^{-\nu},\ m \leq l \leq M,$$

mit $1 \leq a_{-1} \leq g-1$, $0 \leq a_{-\nu} \leq g-1,\ \nu \geq 2$, sowie $0 = +g^m\ .00 \cdots 0$. Es gilt

$$|\mathcal{G}(g, t, m, M)| = 2(g-1)g^{t-1}(M - m + 1) + 1.$$

Gleitpunktrundung
Für $z \in [-g^M(1 - g^{-t-1}), g^M(1 - g^{-t-1})]$ wird mit der Standardrundung wie folgt auf eine im Rechner darstellbare Zahl $\overline{z}$ gerundet.

$$\overline{z} = \begin{cases} 0,\ \text{falls}\ |z| < g^m, \\ \mathrm{sgn}(z) rd(|z|\, g^{-p}) g^p,\ \text{falls}\ |z| \geq g^m,\ p = [\log_g |z|] + 1 \end{cases}.$$

Fehlerabschätzungen
Für $\mathcal{G}(g, t, m, M)$ seien die Grundoperationen $\times$ gemäß der Standardrundung implementiert. Dann gilt

- $|z - \overline{z}| \leq 1/2 g^{-t+1} |z|$ für alle $z \in \mathbb{R}$,
- $|x \times y - (x \otimes y)| \leq |x \times y|\, 1/2 g^{-t+1}$ für alle $x, y \in \mathcal{G}$. Speziell für $g = 2$ gilt $|x \times y - (x \otimes y)| \leq |x \times y|\, g^{-t}$.

Es gibt die Variablentypen `float`, `double` und `long double`, die für Gleitpunktzahlen bestimmt sind. Die dabei verwendete Speicherung ist vom jeweiligen Rechner abhängig. Wir beziehen uns hier auf die Suns. Die wichtigen Informationen zu den Gleitpunktzahlen erhält man über die unten angegebenen Variablen, wobei aber der Header-file `<float.h>` angegeben sein muß. Gleitpunktzahlen erkennt man an der Schreibweise entweder mit einem Dezimalpunkte `.` oder dem Exponentenzeichen `e`. Ein angehängtes `f` bzw. `l` vereinbart `float` bzw. `long double` Zahlen. Die meisten mathematischen Funktionen geben `double` Werte zurück.

	Allgemeines	
`FLT_RADIX`	2	Basis des Zahlsystems
`FLT_ROUNDS`	1	Rundungsart
		1 ist IEEE Rundung
	float	
`sizeof(float))`	4 bytes	Speicherplatzgröße
`FLT_DIG`	6	Genauigkeit dezimal
`FLT_EPSILON`	1.1920929e-07	kleinstes x mit $1.0 + x \neq 1.0$
`FLT_MANT_DIG`	24	Mantissenlänge
`FLT_MIN_EXP`	-125	minimaler Exponent
`FLT_MAX_EXP`	128	maximaler Exponent
	double	
`sizeof(double))`	8 bytes	Speicherplatzgröße
`DBL_DIG`	15	Genauigkeit dezimal
`DBL_EPSILON`	2.220446049250313e-16	kleinstes x mit $1.0 + x \neq 1.0$
`DBL_MANT_DIG`	53	Mantissenlänge
`DBL_MIN_EXP`	-1021	minimaler Exponent
`DBL_MAX_EXP`	1024	maximaler Exponent
	long double	
`sizeof(long double))`	16bytes	Speicherplatzgröße
`LDBL_DIG`	33	Genauigkeit dezimal
`LDBL_EPSILON`	1.925929944387236e-34	kleinstes x mit $1.0 + x \neq 1.0$
`LDBL_MANT_DIG`	113	Mantissenlänge
`LDBL_MIN_EXP`	-16381	minimaler Exponent
`LDBL_MAX_EXP`	16384	maximaler Exponent

5.2 Ein- und Ausgabe

printf("*text1 %speztyp text2*",*var*); *var* wird gemäß der Angaben *spez* und *typ* gedruckt, *text1* und *text2* erscheinen vor und nach der Zahlenausgabe. Es können auch mehrere Variablen ausgegeben werden, zu jeder Variablen muß eine `%` Vereinbarung vorhanden sein. Die Funktion ist vom Typ `int` und liefert die Zahl der gedruckten Zeichen zurück.

Dabei sind die Typen wie folgt festgelegt.

typ		
Format	Variablentyp	Darstellung
`%f`	`float, double`	mit `.`
`%e`	`float, double`	mit `e`
`%g`	`float, double`	wie `%e`, wenn Exponent < -4, sonst wie `%f`
`%Lf %Le %Lg`	`long double`	wie oben, aber für long double

Die *spez* kann fortgelassen werden, dann tritt eine Voreinstellung ein. Zu den reservierten Spalten müssen auch `.` und Vorzeichen gerechnet werden.

spez	
	6 Nachkommastellen
zahl1`.`*zahl2*	*zahl1* reservierten Spalten und *zahl2* Nachkommastellen

scanf("%*typ*", &*var*); Es wird gemäß der Vereinbarung in *typ* eingelesen, die Variable *var* ist damit belegt. Es können auch mehrere Zahlen eingelesen werden, für jede Variable muß eine `%` Vereinbarung gesetzt werden. Die Funktion ist vom Typ `int` und liefert die oben angegebenen Werte zurück.

typ		
Format	Variablentyp	verlangte Darstellung
`%f %e %g`	float	dezimal mit `.` oder `e`
`%lf %le %lg`	double	dezimal mit `.` oder `e`
`%Lf %Le %Lg`	long double	dezimal mit `.` oder `e`

Die besondere Behandlung von `l` und `L`, die bei der Eingabe notwendig ist, kann auch bei der Ausgabe verwendet werden.

Listing 18: klassischer Fehler (1) (classicalerror1.c)

```c
#include <stdio.h>
        /* Ein klassischer Fehler */
int main()
{
        double x;
        printf("Eingabe x (double): ");
        scanf("%f", &x);
        printf("%17.16f\n", x);
        return 0;
}
```

Das nächste Programm liefert den beliebten `Segmentation fault`.

Listing 19: klassischer Fehler (2) (classicalerror2.c)

```c
#include <stdio.h>
        /* den Klassiker behoben,
                  * auf Kosten eines alten Bekannten */
int main() {
        double x;
        printf("Eingabe x (double): ");
        scanf("%lf", x);
        printf("%17.16f\n", x);
return 0;
}
```

In beiden Fällen wird man aber (hoffentlich) vom Compiler eine Warnung erhalten.

explizite Typumwandlung (Casting):

***var* = (*typ*) *expression*;** Der Ausdruck *expr* wird ausgewertet und in den angegebenen Typ *typ*, der dem von *var* entspricht, umgewandelt.

Aliaslisten:

enum *var* {*al1*, *al2*, . . . , *al3* }; Es wird eine Aliasliste mit p. v. *ali* erzeugt, die mit aufsteigenden ganzzahligen Werten verbunden werden. Beginn ist 0, falls nicht ein anderer Wert durch *al1* = *int* vereinbart wurde. Wenn eine Variable mit `enum` definiert wird, kann sie jeden `int`-Wert annehmen, nicht nur die in `enum` festgelegten.

Listing 20: Enum (boole.c)

```c
#include <stdio.h>

enum boolean {FALSE, TRUE};

int main(void)
{
enum boolean aa, bb, cc;
enum escapes {BELL = '\a', BACKSPACE = '\b',
                              HTAB = '\t', RETURN = '\r',
                              NEWLINE = '\n', VTAB = '\v'};
aa = FALSE;
bb = TRUE;
cc = 4713;        /* Das sollte nicht passieren! */
printf("%d, %c %d, %c %d%c",
                    aa, HTAB, bb, HTAB, cc, NEWLINE);
return 0;
}
```

Output:

```
0,       1,      4713
```

Listing 21: Bisektion (1) (bisect1.c)

```
/* Bisektion 1 (nicht ANSI) */
#include <stdio.h>
#include <math.h>
#include <float.h>

long double sqrtl( long double);
long double fabsl(long double);

int main(void)
{
        long double a = 1.4l, b = 1.5l, c;
        long double eps = 2*LDBL_EPSILON;
        long double f(long double);
        int counter = 0;
        printf("Berechnung von sqrt(2)\n");
        while ( fabsl(b - a) > eps)
        {
                counter++;
                c = (a+b)/2.l;
                if ( f(c) * f(a) < 0.l)
                        b = c;
                        else
                        a = c;
                printf("%20.17Lf, %20.17Lf\n", a, b);
         }
        printf("%35.32Lf, nach %d Schritten\n",
                (a + b)/2.l, counter);
        printf("%35.32Lf (mit sqrtl)\n", \
                sqrtl((long double) 2.));
        return 0;
}

long double f(long double x)
{ return x*x - 2.0l;
}
```

Listing 22: Bisektion (2) (bisect2.c)

```
/* Bisektion 2  nicht ANSI */
#include <stdio.h>
#include <math.h>
#include <float.h>
#define LODO long double
LODO sqrtl(LODO), fabsl(LODO);
int main(void)
{ LODO a = 1.4l, b = 1.5l, c, fa, fb, fc;
        LODO half = 0.5l, eps = 2*LDBL_EPSILON;
        LODO f(LODO);
        int counter = 0, maxstep =200;
        printf("Berechnung von sqrt(2)\n");
        fa = f(a);
        fb = fa+100.;
        while ( ( fabsl(fb - fa ) > eps) ||
                        ( fabsl(b - a) > eps) ) {
                counter++;
                if ( counter > maxstep )
                        break;
                c = (a+b)*half;
                fc = f(c);
                if ( fc * fa < 0.l) {
                        b = c;
                        fb = fc;
                } else {
                        a = c;
                        fa = fc; }
        }
        printf("%34.32Lf, nach %d Schritten\n",
                (a + b)*half, counter);
        printf("%34.32Lf (mit sqrtl)\n", sqrtl(2.l));
        return 0; }

LODO f(LODO x)
{ return x*x - 2.0l;}
```