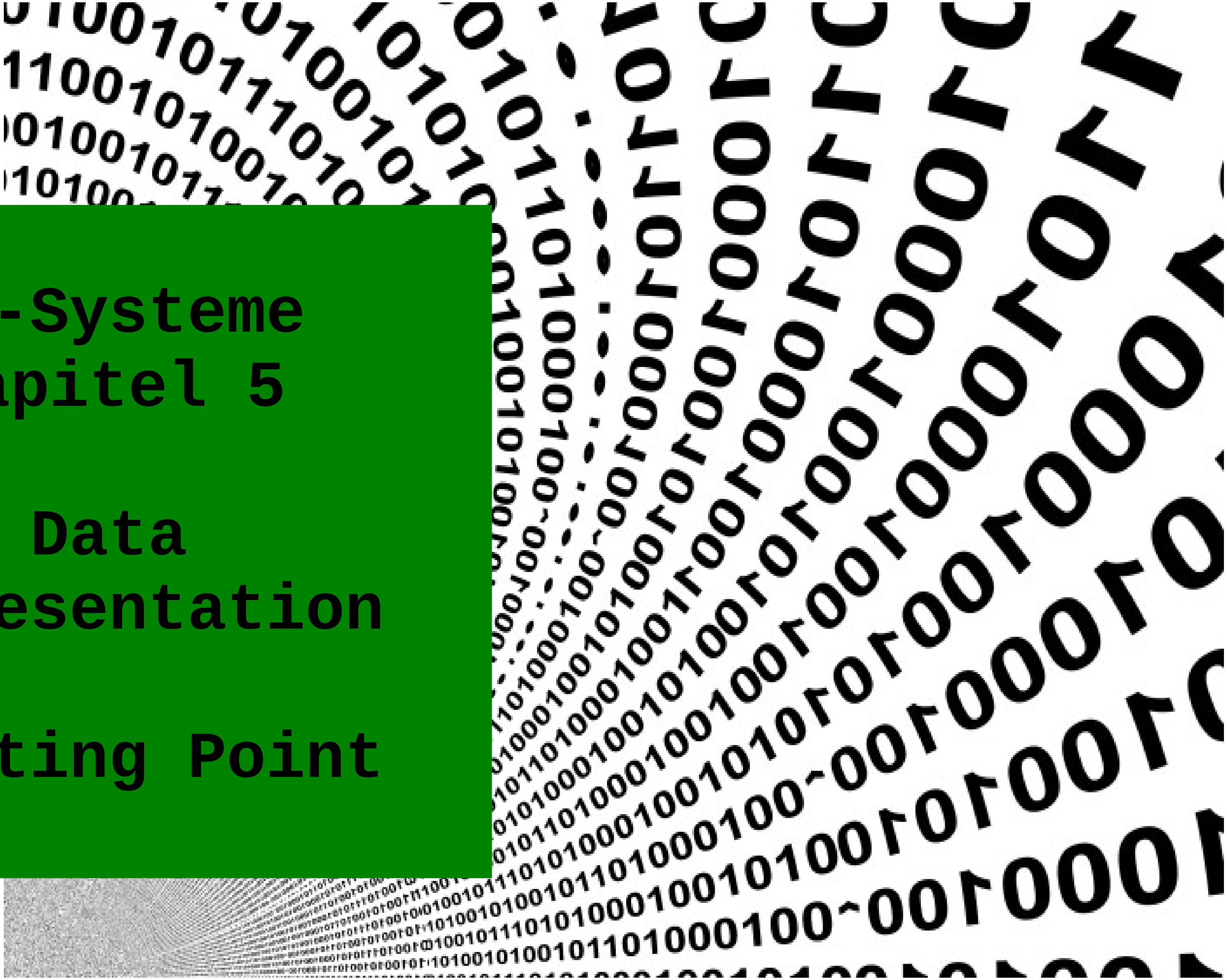


DS-Systeme
Kapitel 5

Data Representation

Floating Point

Inhaltsverzeichnis

Darstellung von Gleitpunktzahlen nach dem IEEE 754-Standard

IEEE 754-Standard für **float** und **double**

bei Festpunktzahlen

Jede reelle Zahl kann in der folgenden Form angegeben werden.

$$\textbf{Mantisse} * \textbf{Basis}^{\textbf{Exponent}}$$

Z.B.: $2.3756 * 10^3$

Der Exponent ist immer ganzzahlig.

Diese Form wird auch in Computern mit der **Basis 2** verwendet – und der Exponent wird durch den sog. **biased exponent** ersetzt. In beiden Fällen wird die erste Ziffer ungleich 0 vor das Komma platziert und der Eponent entsprechend angepasst.

Die für die Darstellung einer Gleitpunktzahl verwendete Anzahl von Bytes legt fest, ob mit einfacher (**float**) oder mit doppelter Genauigkeit (**double**) gerechnet wird.

*Für den Begriff **Mantisse** findet man in der Literatur auch Namen wie **"Signifikand"** oder engl. **"fraction"**, wobei die beiden letzteren die eine Stelle vor dem Komma /Punkt nicht mitzählen.

Darstellung von Gleitpunktzahlen nach dem IEEE 754-Standard

IEEE 754-Standard für **float** und **double**

IEEE 754-Standard geht von normalisierten Gleitpunktzahlen aus. "Normalisierung" bedeutet hier, dass die ursprüngliche Zahl so verändert wird, dass der Dezimalpunkt in der binären Darstellung hinter der ersten 1 steht. Der Exponent muss entsprechend angepasst werden.

Beispiel:

$$10.5_{10} * 10^{0} = 1010.1_{2} * 2^{0}$$
$$= 1.05_{10} * 10^{1} = 1.0101_{2} * 2^{3}$$

Die Dezimalzahl

$$17.625 = 1 \cdot 10^1 + 7 \cdot 10^0 + 6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}$$

entspricht der binären Zahl:

$$16 \quad + \quad 1 \quad + \quad 1/2 \quad + \quad 1/8$$
$$= 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 10001.101 \cdot 2^0$$

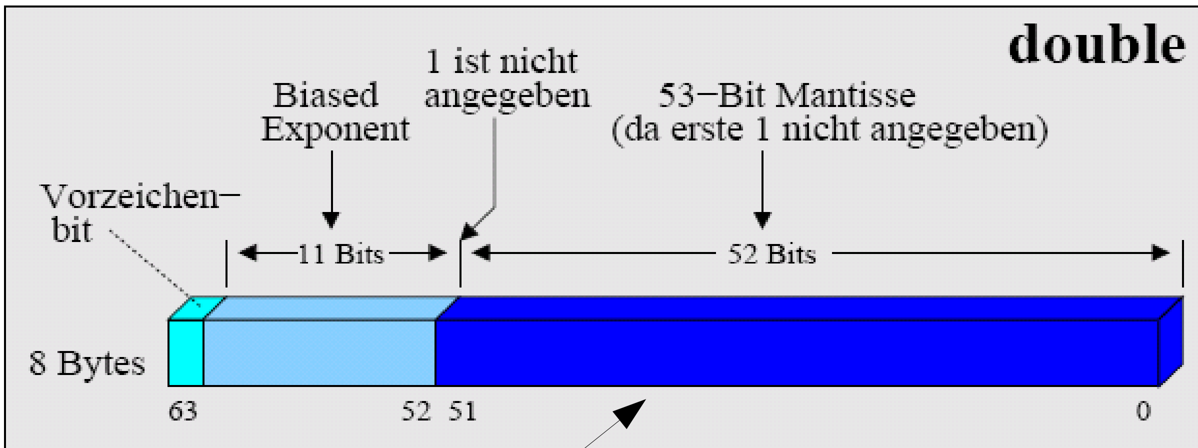
Die entsprechende normalisierte Form erhält man, indem man den Dezimalpunkt hinter die erste signifikante Ziffer „schiebt“ und den Exponenten entsprechend anpasst:

$$1.0001101 \cdot 2^4$$

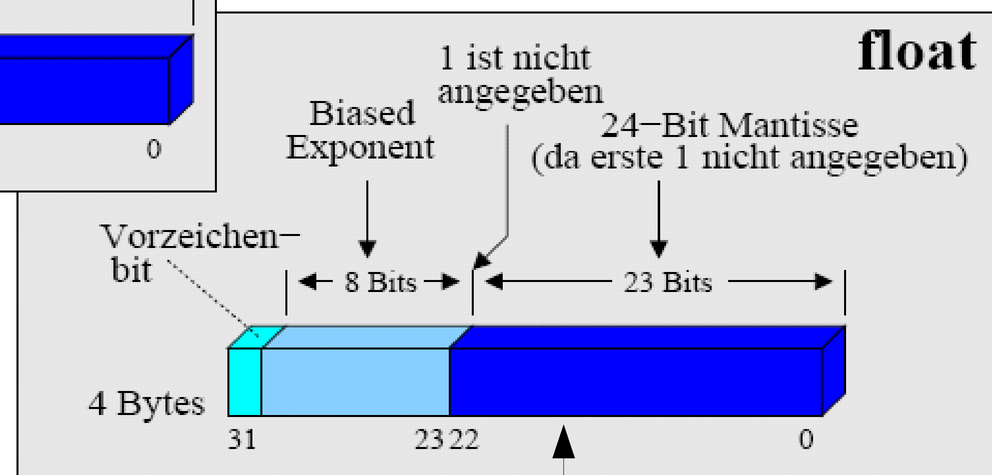
Darstellung von Gleitpunktzahlen nach dem IEEE 754-Standard

IEEE 754-Standard für **float** und **double**

Zur Darstellung der Datentypen **float** und **double** verwenden C, C++, Java u.a. den IEEE 754-Standard:



52 Bits = Signifikand
(engl. significant
bzw. fraction)
53 Bits = Mantisse



23 Bits = Signifikand
(engl. significant
bzw. fraction)
24 Bits = Mantisse

Darstellung von Gleitpunktzahlen nach dem IEEE 754-Standard

IEEE 754-Standard für **float** und **double**

In der Mantisse steht durch die normalisierte Form das höchstwertige "Einser-Bit" immer links vom gedachten Dezimalpunkt (außer für 0.0). Beim IEEE 754-Standard wird dieses Bit nicht gespeichert.

Der Exponent ist eine Ganzzahl, welche – nach Addition eines **Bias** – ohne Vorzeichen dargestellt wird.

Durch diese **Bias**-Addition wird für den Exponent keine Vorzeichenrechnung benötigt.

Der Wert vom **Bias** hängt vom Genauigkeitsgrad ab:

float (mit 4 Bytes, 8 Bits für Exponent): **bias** = 127
double (mit 8 Bytes, 11 Bits für Exponent): **bias** = 1023

Das Vorzeichenbit zeigt das Vorzeichen der Mantisse, die immer als Betragswert, auch im negativen Fall nicht als Komplement, dargestellt wird.

Formel zur Darstellung einer Gleitpunktzahl nach IEEE 754

IEEE 754-Standard für **float** und **double**

Formel zur Darstellung einer Gleitpunktzahl im IEEE-Format:

$$(-1)^S \cdot \underbrace{(2^{B-bias})}_{EXPONENT} \cdot \underbrace{(1.f^N \ldots\ldots f^0)}_{SIGNIFICAND}$$

SIGNIFICAND
N= 23 (`float`= 24 Stellen), N= 52 (`double`= 53 Stellen)

EXPONENT B = Biased Exponent (zu speichernder Exponent)
bias = 127 (`float`), bias = 1023 (`double`)

SIGN S = 0 (positiv); S = 1 (negativ)

Beispiel:

$(17.625)_{10} = (1.0001101 \bullet 2^4)_2$

```
     |0|10000011|00011010000000000000000|
Pos. 31         /                       0
          Biased Exponent ergibt sich als:
                             bias =     0111 1111 = 127
          + wirklicher Exponent =       0000 0100 =   4
                                        ---------------
                                        1000 0011 = 131
```

Beispiel 1: Zahl besteht aus Summe von 2er-Potenzen

IEEE 754-Standard für **float** und **double**

Ausführliches Beispiel 1: $17.625_{10} = ?_2$

Vor- bzw. Nachkommastellen werden getrennt berechnet.

Vorkommastellen

```
17 : 2 = 8 Rest 1
 8 : 2 = 4 Rest 0
 4 : 2 = 2 Rest 0
 2 : 2 = 1 Rest 0
 1 : 2 = 0 Rest 1
```

$17_{10} = 10001_2$

Nachkommastellen

```
0.625 * 2 = 1.25  Übertrag 1
0.25  * 2 = 0.5   Übertrag 0
0.5   * 2 = 1.0   Übertrag 1
```

$0.625_{10} = 0.101_2$

Ergebnis: $17.625 = 17_{10} + 0.625_{10} = 10001_2 + 0.101_2 = \mathbf{10001.101_2}$

Beispiel 1: Zahl besteht aus Summe von 2er-Potenzen

IEEE 754-Standard für **float** und **double**

Ausführliches Beispiel 1: $17.625_{10} = ?_2$

Darstellung im Computer

Das Komma in der Dualzahl wandert hinter die erste 1. Diese 1 wird im Rechner nicht mehr dargestellt.

$10001.101 * 2^0 = 1.0001101 * 2^4$ D.h. der neue Exponent zur Basis 2 ist 4.

Aufaddieren des **Bias** (bei float 127)

```
    4        00000100
+ 127      + 01111111
-----      ----------
  131        10000011
```

| 0 | 1 0 0 0 0 0 1 1 | 0 0 0 1 1 0 1 0 0 … 0 |

Vorzeichenbit — biased exponent — significant bits (Nachkommastellen)

Beispiel 1: Zahl besteht nicht aus Summe von 2er-Potenzen

IEEE 754-Standard für **float** und **double**

Ausführliches Beispiel 2: $-22.05_{10}=?_2$

Vor- und Nachkommastellen werden getrennt berechnet.

Vorkommastellen

```
22 : 2 = 11 Rest 0
11 : 2 =  5 Rest 1
 5 : 2 =  2 Rest 1
 2 : 2 =  1 Rest 0
 1 : 2 =  0 Rest 1
```

Nachkommastellen

```
0.05 * 2 = 0.1 Übertrag 0
0.1  * 2 = 0.2 Übertrag 0
0.2  * 2 = 0.4 Übertrag 0
0.4  * 2 = 0.8 Übertrag 0
0.8  * 2 = 1.6 Übertrag 1
0.6  * 2 = 1.2 Übertrag 1
0.2  * 2 = 0.4 ...
```

Ab hier wiederholen sich die letzten 4 Zeilen.

$-22_{10}=-10110_2$

$-0.05_{10}=-0.00\overline{0011}_2$

Ergebnis: $-22.05=-22_{10}-0.05_{10}=-10110_2-0.00\overline{0011}_2=\mathbf{-10110.00\overline{0011}_2}$

Beispiel 1: Zahl besteht nicht aus Summe von 2er-Potenzen

IEEE 754-Standard für **float** und **double**

Ausführliches Beispiel 2: $-22.05 = \mathbf{-10110.00\overline{0011}_2}$

Darstellung im Computer

Das Komma in der Dualzahl wandert hinter die erste 1. Diese 1 wird im Rechner nicht mehr dargestellt.

$$-10110.00\overline{0011}_2 * 2^0 = -1.011000\overline{0011}_2 * 2^4$$

D.h. der neue Exponent zur Basis 2 ist 4.

Aufaddieren des **Bias** (bei float 127)

```
    4         00000100
+ 127       + 01111111
-----       ----------
  131         10000011
```

| 1 | 1 0 0 0 0 0 1 1 | 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 1 … 1 1 0 |

Vorzeichenbit

biased exponent

significant bits (Nachkommastellen)

Runden der letzten Stelle der Signifikanden

IEEE Rounding-Modes: Round to Nearest

Guard-
Round-
Sticky

Mantisse	G	R	S

Eingabe	Form bei Tie	Rundung	Ergebnis M
Mantisse|000		Same	M = Mantisse
Mantisse|001		Down	M = Mantisse
Mantisse|010		Down	M = Mantisse
Mantisse|011		Down	M = Mantisse
Mantisse|100 (Tie)	…0|100	Down	M = Mantisse
	…1|100	Up	M = Mantisse + 1
Mantisse|101		Up	M = Mantisse + 1
Mantisse|110		Up	M = Mantisse + 1
Mantisse|111		Up	M = Mantisse + 1

Die drei Bits in der Spalte "Eingabe" beziehen sich auf zusätzlich zu der **float**- bzw. **double**-Darstellung errechnete Bits.

Das erste dieser drei ist das **Guard-Bit**, das darüber entscheidet, ob noch aufgerundet wird oder nicht.

Runden der letzten Stelle der Signifikanden

Bezogen auf das letzte Beispiel, $-22.05 = \mathbf{-10110.00\overline{0011}_2}$
sieht das Ende dann zunächst folgendermaßen aus:

| 1 | 1 0 0 0 0 0 1 1 | 0 1 1 0 0 0 0 0 1 1 0 0 1 1 0 0 1 … 1 1 0 |

Da sich in diesem Beispiel die Ziffern 0011 immer wiederholen, ergeben sich als zusätzliche Bits die Ziffern 011.

... 0 0 1 1 0 0 1 1 0 0 1 1 0 |

0 1 1

Drei zusätzlich berechnete Bits

Da das **Guard-Bit** auf **0** steht, muss hier nichts weiter unternommen werden und die o.a. Darstellung ist korrekt.

Aufgabe:

Berechnen Sie die interne **float**-Darstellung der Zahl **0.3**.

Not a Number (NaN) und infinity (INFINITY)

Darstellung bzgl. IEEE 754-Standard – Datentyp **float**

INFINITY bzw. -INFINITY

interne Darstellung:

maximal mögl. Wert des **biased exponents** = 255_{10} bzw. 11111111_2
zusätzlich alle Mantissen-Bits 0
Vorzeichen positiv (0) entspricht INFINITY
Vorzeichen negativ (1) entspricht -INFINITY

Entsteht z.B. bei Division durch 0.0.

NAN

interne Darstellung:

biased exponent = 255_{10} bzw. 11111111_2
zusätzlich mindestens ein Mantissen-Bit 1
(es ist möglich, dass hier sogar alle Mantissen-Bits 1 werden)
das Vorzeichen-Bit kann 0 oder 1 sein

Hinweis:

NAN und INFINITY treten nur bei Gleitkomma-Datentypen auf.

INFINITY

```c
#include <stdio.h>
#include <math.h> // isinf()
#include <float.h>
#include <limits.h>

int main()
{
    printf("isinf(NAN)         = %2d\n", isinf(NAN));
    printf("isinf(INFINITY)    = %2d\n", isinf(INFINITY));
    printf("isinf(0.0)         = %2d\n", isinf(0.0));
    printf("isinf(DBL_MIN/2.0) = %2d\n", isinf(DBL_MIN/2.0));
    printf("isinf(1.0)         = %2d\n", isinf(1.0));
    printf("isinf(exp(800))    = %2d\n", isinf(exp(800)));

    return 0;
}
```

Ausgabe:

```
isinf(NAN)         =  0
isinf(INFINITY)    =  1
isinf(0.0)         =  0
isinf(DBL_MIN/2.0) =  0
isinf(1.0)         =  0
isinf(exp(800))    =  1
```

Not a Number (NaN)

```c
#include <stdio.h>
#include <math.h>
#include <float.h>

int main(void)
{
    printf("isnan(sqrt(-1)))   = %d\n", isnan(sqrt(-1)));
    printf("isnan(NAN)         = %d\n", isnan(NAN));
    printf("isnan(INFINITY)    = %d\n", isnan(INFINITY));
    printf("isnan(0.0)         = %d\n", isnan(0.0));
    printf("isnan(DBL_MIN/2.0) = %d\n", isnan(DBL_MIN/2.0));
    printf("isnan(0.0 / 0.0)   = %d\n", isnan(0.0/0.0));
    printf("isnan(Inf - Inf)   = %d\n", isnan(INFINITY - INFINITY));
}
```

Ausgabe:

```
isnan(sqrt(-1.0))  =  1
isnan(NAN)         =  1
isnan(INFINITY)    =  0
isnan(0.0)         =  0
isnan(DBL_MIN/2.0) =  0
isnan(0.0 / 0.0)   =  1
isnan(Inf - Inf)   =  1
```

Mathematische Sichtweise auf die IEEE 754-Darstellung

Formel für reelle Zahlen

Sei $b \in \mathbb{N}$ eine Basis, $b > 1$ und $z \in \mathbb{R}$ mit n Vorkommastellen, dann ist der Betrag $|z_n| <= b-1$. Dann ist folgende Darstellung eindeutig:

$$z = (-1)^v \sum_{i=-\infty}^{n-1} z_i b^i \text{ mit } \begin{cases} v \in \{0,1\}, v=0 \text{ für } z=0 \\ z_i \in \sum_b, i=0,\ldots,n-1 \\ z_i \leq b-1 \text{ für unendlich viele } i<n \end{cases}$$

Schreibweise: $z = \pm z_{n-1} \ldots z_0, z_{-1} \ldots$

Darstellung reeller Zahlen auf dem Rechner:

Einschub Festkommazahlen:

k = #Stellen gesamt, **n** = #Vorkommastellen, **k-n** = #Nachkommastellen

$$z = (-1)^v \sum_{i=-(k-n)}^{n-1} z_i b^i$$

Schreibweise:

$$z = \pm z_{n-1} \ldots z_0, z_{-1} \ldots z_{-(k-n)}$$

Mathematische Sichtweise auf die IEEE 754-Darstellung

Gleitpunktzahlen

m = Mantisse, **e** = Exponent

$$z=(-1)^v m \cdot b^e, v \in \{0,1\}$$

Diese Form ist nicht eindeutig, z.B.: -23.45 * 10^2 = -2345 * 10^0

Normalisierte Gleitpunktzahlen:

k = #Stellen der Mantisse

$$z=(-1)^v \left(\sum_{i=0}^{k-1} m_i b^{-i} \right) b^e, v \in \{0,1\}, m_i \in \sum_b, m_0 \neq 0 \quad (*)$$

Beispiel mit **k** = 4:

24.68 * 10^0 = (2 * 10^0 + 4 * 10^{-1} + 6 * 10^{-2} + 8 * 10^{-3}) * 10^1
= 2.468 * 10^1 (normalisiert)

Definition 1:

N = Normalisierte Gleitpunktzahl

$\mathbf{M_N(b, k, e_{min}, e_{max})}$ bezeichnet die Menge der durch die Formel (*) darstellbaren normalisierten Gleitpunktzahlen.

Beispiel mit $\mathbf{M_N(2, 3, -1, 1)}$:

Exponent / Mantisse	2^{-1}	2^0	2^1
1.00	0.5	1	2
1.01	0.625	1.25	2.5
1.10	0.75	1.5	3
1.11	0.875	1.75	3.5

dezimal

Mathematische Sichtweise auf die IEEE 754-Darstellung

Definition 2:

Die Zahl **z** = 0 wird durch $\mathbf{m_i}$ = 0 für alle i = 0, … , k-1 und $\mathbf{e_{min}}$ definiert.
Alle anderen Zahlen mit der Darstellung

$$z=(-1)^v\left(\sum_{i=0}^{k-1} m_i b^{-i}\right) b^{e_{min}}, v \in \{0,1\}, m_i \in \sum_b, m_0=0$$

heißen sub- bzw. denormale Zahlen.

Die Menge $$M(b,k,e_{min},e_{max}) := M_N(b,k,e_{min},e_{max}) \cup \{0\} \cup M_S(b,k,e_{min},e_{max})$$

heißt die Menge der Gleitpunkt- bzw. Maschinenzahlen.

Denormale bzw. subnormale Zahlen

Testprogramm:

```
#include <stdio.h>

int main()
{
    // Darstellung der Null
    unsigned long long \
    ull = 0b0000000000000000000000000000000000000000000000000000000000000000;
    double * pd = (double *)&ull;

    // Darstellung subnormaler Zahlen
    // Hier kann man beliebig viele Bits der Mantisse auf 1 setzen
    // Exponent muss komplett aus Nullen bestehen.
    printf("%.30le\n", *pd);
    ull = 0b0000000000000000010000000000000000000100000000000010000000000001;
    pd = (double *)&ull;
    printf("%.30le\n", *pd);
    return 0;
}
```

Aufgabe:

Geben Sie die Bit-Darstellung der kleinsten und der größten normalisierten **float**-Zahl im IEEE 754-Standard an.

Mathematische Sichtweise auf die IEEE 754-Darstellung

Auf dem Rechner gilt wegen Basis 2 immer $m_0 = 1$. Dieses führende Bit wird nicht gespeichert, d.h. bei k-stelliger Mantisse gilt

$$z = (-1)^{\nu} \left(1 + \sum_{i=1}^{k} m_i 2^{-i}\right) 2^e, \quad \nu, m_i \in \{0,1\}, \text{ also } m = (1.z_1 \dots z_k).$$

Werden für den Exponent l Stellen verwendet, gilt

biased exponent

$$e = \sum_{i=0}^{l-1} e_i 2^i - (2^{l-1} - 1), \quad e_i \in \{0,1\}, i = 0, \dots, l-1 \text{ und damit } e \geq e_{min} := -(2^{l-1} - 1)$$

und

$$e \leq e_{max} := \sum_{i=0}^{l-1} 2^i - (2^{l-1} - 1) = 2^l - 1 - (2^{l-1} - 1) = 2^l - 2^{l-1} = 2^{l-1}$$

Mit $e = e_{max}$ und $m_i = 0$ für alle $i = 1, \dots, k$ werden die Werte **+INFINITY** und **-INFINITY** repräsentiert.

Mit $e = e_{max}$ und $m_i \neq 0$ für mindestens ein i aus $\{1, \dots, k\}$ wird **NaN** (not a number) dargestellt.

Ist $e = e_{min}$ und $m_i \neq 0$ für mindestens ein i aus $\{1, \dots, k\}$ handelt es sich um eine **subnormale Zahl**.*

Legende:

- e: Exponent
- l: #Ziffern des Exponenten
- m: gesamte Mantisse
- m_i: einzelne Ziffern der Mantisse
- v: Vorzeichen
- z: vollständige Zahl
- z_i: einzelne Ziffern der Zahl

* Ausnahme: Die **0 (alle** $m_i = 0$**)** ist auch subnormal.