Technische Universität München
Institut für Informatik
Prof. Dr. Thomas Huckle
Sebastian Wolf
Shuo Sun
Hayden Liu Weng

SoSe 2023

Numerisches Programmieren, Übungen

Musterlösung 1. Übungsblatt: Zahlendarstellung, Rundungsfehler

1) Umrechnung von Zahlen

a) Schreiben Sie die ganzen Zahlen 19, 47 und 511 in binärer und hexadezimaler Darstellung!

b) Schreiben Sie die Brüche $-\frac{3}{8}$ und $\frac{1}{10}$ als Binärzahl!

Lösung:

a) Zahldarstellung in allgemeinem System: $\sum_{i=0}^{N} r_i \text{Basis}^i$ mit $r_i \in \{0, 1, \ldots, \text{Basis} - 1\}$

	dezimal	binär	trinär	hexadezimal
Basis	10	2	3	16
darzust. Zahl	19	10011	201	13
darzust. Zahl	47	101111	1202	2f
darzust. Zahl	511	111111111	200221	1ff

b) Schriftliches Dividieren der Brüche im binären System analog zum dezimalen:

$-\frac{3}{8} = -11_2 : 1000_2$ (binär) :

$$\begin{array}{rrrrrrrrrrrr} - & 1 & 1 & 0 & 0 & : & 1 & 0 & 0 & 0 & = & -0.011 \\ -) & 1 & 0 & 0 & 0 & & & & & & & \\ \hline & & 1 & 0 & 0 & 0 & & & & & & \\ -) & & 1 & 0 & 0 & 0 & & & & & & \\ \hline & & & & & 0 & & & & & & \end{array}$$

$\frac{1}{10} = 1_2 : 1010_2$ (binär) :

$$\begin{array}{rrrrrrrrrrrr} & 1 & 0 & 0 & 0 & 0 & : & 1 & 0 & 1 & 0 & = & 0.0001\overline{1100} \text{ bzw. } 0.\overline{00011} \\ -) & & 1 & 0 & 1 & 0 & & & & & & & \\ \hline & & & 1 & 1 & 0 & 0 & & & & & & \\ -) & & & 1 & 0 & 1 & 0 & & & & & & \\ \hline & & & & 1 & 0 & 0 & 0 & & & & & \\ -) & & & & 1 & 0 & 1 & 0 & & & & & \\ \hline & & & & & 1 & 0 & 1 & 0 & & & & \\ -) & & & & & 1 & 0 & 1 & 0 & & & & \\ \hline & & & & & & 0 & 0 & 0 & 0 & & & \end{array}$$

$-\frac{3}{8}$ $\frac{1}{10}$ 转2进制

2) Assoziativgesetz

Eine Eigenschaft, die bei Gleitkommazahlen leider verloren wird, ist das Assoziativgesetz. Betrachten Sie eine binäre Gleitkomma-Darstellung mit 4 signifikanten Stellen.

Beispiele:

Zahl	Anzahl signifikanter Stellen
1000000.0_2	8
$1 \cdot 2^6$	1
1000.1_2	5
-0.01011_2	4
10.10_2	4

Berechnen Sie im gegebenen Format die Werte

- $(-8 + 11) + 0.75$ und
- $-8 + (11 + 0.75)$.

Runden Sie dabei nach jedem Rechenschritt. Was ist zu beobachten?

Lösung: Die Zahlen in binär: Damit gilt:

dezimal	binär
-8	-1000_2
11	1011_2
0.75	0.11_2

$$
\begin{aligned}
(-8 + 11) + 0.75 &= (-1000_2 + 1011_2) + 0.11_2 = (11_2) + 0.11_2 \\
&= 11.11_2 = \boxed{3.75} \quad \text{(kein Runden nötig)} \\
-8 + (11 + 0.75) &= -1000_2 + (1011_2 + 0.11_2) = -1000_2 + (1011.11_2) \\
&\overset{\text{Runden!}}{=} -1000_2 + (1100_2) \\
&= 100_2 = \boxed{4}
\end{aligned}
$$

Das Ergebnis ist offensichtlich abhängig von der Reihenfolge der abgearbeiteten Operationen. Diese Rundungsfehler können auch zu sehr großen Fehlern führen. Dabei sei auf die Aufgabe zur Stabiliätsanalyse auf dem nächsten Blatt verwiesen.

3) Gleitkomma-Zahlen

In dieser Aufgabe soll Schritt für Schritt eine reelle Zahl in eine Maschinenzahl umgewandelt werden. Die umzuwandelnde Zahl lautet $-\frac{11}{10}$.

a) Schreiben Sie die Zahlen zuerst in eine andere, standardisierte Form folgender Gestalt um! An erster Stelle steht das Vorzeichen. Dann folgt der Betrag der Zahl in binärer Exponentialdarstellung, beginnend mit »1, . . . «. Es geht weiter mit den Nachkommastellen, einem Malpunkt und abschließend steht eine Zweierpotenz (Beispiel: $-1,11001 \cdot 2^{-56}$).

b) Wandeln Sie die Ergebnisse von Teilaufgabe a) in Binärcode um! Dazu verwenden Sie den im Folgenden spezifizierten 32-Bit IEEE-Standard:

$(-8+11)+0.75$ 4位有效数字

$-8: -1000$

$11: 1011$ (8 2 1)

$0.75 = 0.11$

$(-1000+1011)+0.11$

即: $11+0.11$

$= 11.11$

$-1000+(1011+0.11)$

$= -1000 + 1011.11$

Runden

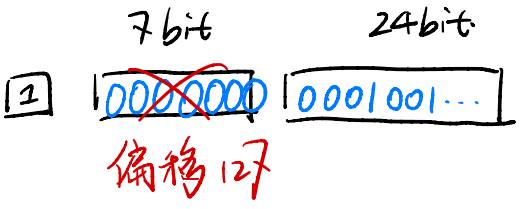
$= -1000 + 1100$

$= 100_2$

$-\frac{11}{10}$ $\quad 11: 1011_2 \quad 10: 1010_2$

$$
\begin{array}{r}
1.0001 \\
1010\overline{)1.0001100110001} \\
1010 \\
\hline
10000 \\
1010 \\
\hline
1100 \\
1010 \\
\hline
10000 \\
1010 \\
\hline
1100 \\
1010 \\
\hline
10000
\end{array}
$$

$-1.\overline{0001}\times 2^{0}$



- Das erste Bit bestimmt das Vorzeichen: Eine Null bedeutet positive Zahl, eine Eins bedeutet negative Zahl.
- Die nächsten 8 Bits sind für den Exponenten reserviert. Die gespeicherte Binärzahl entspricht dem Exponenten plus 127. Die Bit-Kombinationen 00000000 (entspräche einem Exponenten von -127) und 11111111 (entspräche $+128$) sind allerdings für spezielle Werte reserviert (0, ∞, NaN, ...).
- Die letzten 23 Bits dienen der Speicherung der Nachkommastellen. Dabei wird wie in Teilaufgabe a) von einer Normalisierung mit führender Eins ausgegangen (die nicht gespeichert werden muss). Genügen 23 Bits Genauigkeit für die Mantisse nicht, so wird zum nächstgelegenen darstellbaren Wert gerundet.
- Es gäbe noch viel über IEEE zu sagen, was aber nicht zur Lösung dieser Aufgabe notwendig ist. Weitere Informationen finden Interessierte zum Beispiel im Artikel »What Every Computer Scientist Should Know about Floating-Point Arithmetic« von David Goldberg.

c) Wir haben nun eine Zahl gesehen, welche mit dem in Teilaufgabe b) beschriebenem IEEE-Standard nicht exakt darstellbar ist. Wir wollen nun den Fehler einer solchen Rundung berechnen. Zur Vereinfachung der Rechnung werden wir uns die Zahl $x = 1 + 2^{-30}$ anschauen. Wie sieht die gerundete Zahl aus? Berechnen sie außerdem den absoluten und relativen Fehler.

d) Wie groß ist die Maschinengenauigkeit ε_{Ma} für den in Teilaufgabe b) beschriebenen IEEE-Standard? Durch welche Größe wird die Maschinengenauigkeit verändert? Welche andere Größe gibt es und was wird hierdurch reguliert?

e) Wie unterscheidet sich der Zahlenbereich eines IEEE **float** und eines **signed int**? Geben Sie hierfür den Zahlenbereich der Gleitkommazahlen getrennt für positive und negative Zahlen an! Geben Sie außerdem an wie sich die Schrittweite der Diskretisierung verhält.

f) Geben sie die erste Ganzzahl an die mithilfe des 32-Bit IEEE-Standards für Gleitkommazahlen nichtmehr exakt darstellbar ist. Welche Probleme entstehen hieraus für Programmiersprachen, welche nur mit Gleitkommazahlen arbeiten und keine Integerzahlen besitzen (z.B. Javascript).

Lösung:

a) Schriftliches Dividieren (wie in Aufg. 1): $-\frac{11}{10} = -1011_2 : 1010_2$ (binär) :

$$
\begin{array}{rccccccccccc}
 & 1 & 0 & 1 & 1 & : & 1 & 0 & 1 & 0 & = & -1.000\overline{1100} \quad \text{bzw.} \quad -1.0\overline{0011} \\
\text{-)} & 1 & 0 & 1 & 0 & & & & & & & \\
\hline
 & & & & 1 & 0 & 0 & 0 & 0 & & & \\
\text{-)} & & & & & 1 & 0 & 1 & 0 & & & \\
\hline
 & & & & & & 1 & 1 & 0 & 0 & & \\
\text{-)} & & & & & & 1 & 0 & 1 & 0 & & \\
\hline
 & & & & & & & & 1 & 0 & 0 & 0 \quad 0
\end{array}
$$

Damit erhält man insgesamt: $-1.0\overline{0011} \cdot 2^0$

Alternative:

$$
\begin{array}{lcccc|c}
\frac{11}{10} & \cdot & 1 & = & \frac{11}{10} \geq 1 & 1 \\
 & & & & & . \\
\frac{1}{10} & \cdot & 2 & = & \frac{2}{10} < 1 & 0 \\
\frac{2}{10} & \cdot & 2 & = & \frac{4}{10} < 1 & 0 \\
\frac{4}{10} & \cdot & 2 & = & \frac{8}{10} < 1 & 0 \\
\frac{8}{10} & \cdot & 2 & = & \frac{16}{10} \geq 1 & 1 \\
\frac{6}{10} & \cdot & 2 & = & \frac{12}{10} \geq 1 & 1 \\
\frac{2}{10} & \cdot & 2 & = & \frac{4}{10} < 1 & 0 \\
\frac{4}{10} & \cdot & 2 & = & \frac{8}{10} < 1 & 0 \\
\frac{8}{10} & \cdot & 2 & = & \frac{16}{10} \geq 1 & 1 \\
\frac{6}{10} & \cdot & 2 & = & \frac{12}{10} \geq 1 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots \quad \vdots & \vdots
\end{array}
$$

b) Rundungsregel (korrektes Runden) für eine Zahl $x = (-1)^\nu \cdot 2^e \cdot 1.x_1x_2 \ldots x_{t-1}|x_tx_{t+1}x_{t+2} \ldots$:

Nr.	Fallbed.	Rundungsvorschrift
Einfacher Fall: Abrunden		
1)	$x_t = 0$	$\mathrm{rd}(x) = (-1)^\nu \cdot 2^e \cdot 1.x_1x_2 \ldots x_{t-1}$
Einfacher Fall: Aufrunden		
2)	$x_t = 1 \wedge$ $\lvert x_tx_{t+1}x_{t+2} \ldots \neq \lvert 1000000000 \ldots$	$\mathrm{rd}(x) = (-1)^\nu \cdot 2^e \cdot (1.x_1x_2 \ldots x_{t-1} + 2^{-(t-1)})$
Runden zur näheren geraden Mantissen-Zahl:		
3)	$x_{t-1}\lvert x_tx_{t+1} \ldots = 0\lvert 100000000 \ldots$	$\mathrm{rd}(x) = (-1)^\nu \cdot 2^e \cdot 1.x_1x_2 \ldots x_{t-1}$
4)	$x_{t-1}\lvert x_tx_{t+1} \ldots = 1\lvert 100000000 \ldots$	$\mathrm{rd}(x) = (-1)^\nu \cdot 2^e \cdot (1.x_1x_2 \ldots x_{t-1} + 2^{-(t-1)})$

Erläuterungen zu den verschiedenen Fällen:
Fall 1) und 2) stellen den normalen Fall dar, dass eine reelle Zahl nicht genau in der Mitte zwischen den beiden nächsten Maschinenzahlen liegt; es wird zum näheren Nachbarn gerundet. Fall 3) und 4) bewirken ein Runden mit Mantissenende $x_{t-1} = 0$ für reelle Zahlen, die genau zwischen zwei Maschinenzahlen liegen. In Fall 2) und 4) wird mit $+2^{-(t-1)}$ das letzte Bit um eins erhöht (und eventuell ein Übertrag durchgeführt).

Frage: Wie kann eine Floating Point Einheit auf der CPU unendlich viele Stellen zum Runden ausrechnen (z. B. bei periodischen Nachkommastellen)?
Das ist nicht nötig. Z. B. muss bei der Division lediglich bekannt sein, ob ein x_{t+1} ungleich 0 ist. Diese Information ist über den Rest erhältlich, der ab dem Berechnen von der Stelle x_{t+1} übrig bleibt.

Beispiel: Es stehen zwei Mantissenbits zur Verfügung, also: $1, x_1x_{t-1}|x_tx_{t+1}x_{t-2} \ldots$
Damit sind die folgenden Zahlen exakt darstellbar:

$$\begin{aligned} 1,00_2 &= 1_{10} \\ 1,01_2 &= 1,25_{10} \\ 1,10_2 &= 1,5_{10} \\ 1,11_2 &= 1,75_{10} \end{aligned}$$

Beispiele für alle vier Fälle aus der obigen Tabelle:

Nr.	Binärzahl	Dezimalzahl	ab-/aufrunden
1)	$1,\underline{00}\lvert 011_2$	$1,09375_{10}$	abrunden
2)	$1,00\lvert 111_2$	$1,21875_{10}$	aufrunden
3)	$1,\underline{00}\lvert 100_2$	$1,125_{10}$	abrunden
4)	$1,0\underline{1}\lvert 100_2$	$1,375_{10}$	aufrunden

Abbildung 1 veranschaulicht die vier Fälle auf dem Zahlenstrahl.

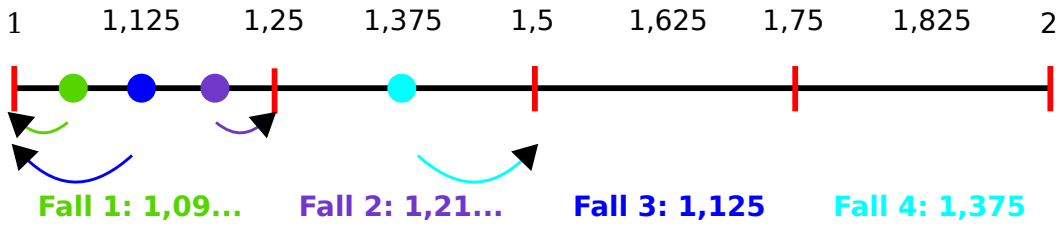


Abbildung 1: Veranschaulichung von Rundungen auf dem Zahlenstrahl

Für unsere Zahl $-\frac{11}{10} = -1.0\overline{0011} \cdot 2^0$ aus Teilaufgabe a) bedeutet das:

Bit-Verwendungszweck	gesetztes Bit für $-1.0\overline{0011} \cdot 2^0$
Vorzeichen (1 Bit) ('-' wird zu '1')	1
Exponent (8 Bits) (0 + 127 = 127)	0 1 1 1 1 1 1 1
Mantisse (23 Bits)	0 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 1

Das letzte Bit der Mantisse ergibt sich nach der Rundungsregel (s.o.).

c) Die Zahl $x = 1 + 2^{-30}$ ist nicht exakt darstellbar (zuwenig Stellen). Die Zahl wird als $\mathrm{rd}(x) = 1$ abgespeichert (vgl. Rundungsregeln).

- Absoluter Fehler: $f_{\text{abs}} := |x - \mathrm{rd}(x)| = 2^{-30}$
- Relativer Fehler: $f_{\text{rel}} := |f_{\text{abs}}/x| < 2^{-30}$

d) Definition: Maschinengenauigkeit = Die größte positive Zahl ε_{Ma}, so dass $1 \oplus \varepsilon_{\text{Ma}} = 1$.

Die kleinste darstellbare Zahl größer als 1 ist $1 + 2^{-23}$ (23 Bits echt für Mantissen-Nachkommastellen frei, da Normierung '1.' nicht extra gespeichert werden muss!). Folglich liegt ε in einer Größenordnung von 2^{-24}.

e) Beim IEEE float ist der Zahlenbereich identisch für positive und negative Zahlen (bis auf das Vorzeichen). Hier können Zahlen zwischen $1 * 2^{\exp_{\min}}$ und $2 - 2^{-m} * 2^{\exp_{\max}}$ dargestellt werden. Bei **float** mit 32Bit gilt (unter Berücksichtigung der Sonderfälle) für den minimalen Exponent $exp_{\min} = -126$, für den maximalen Exponent $\exp_{\max} = 127$ und für die Mantissenlänge $m = 23$. Ein signed **int** hingegen hat bildet den Zahlenbereich von -2^{31} bis $2^{31} - 1$ ab und ist erheblich kleiner. Es gibt aber natürlich nicht mehr **int** Zahlen als **float**. Der Unterschied besteht in der Schrittweite. Bei **int** ist die Schrittweite konstant bei 1 und bei **float** ist sie abhängig vom Exponent e der Zahl $2^{-m} \cdot 2^e = 2^{e-m}$. Der Abstand zwischen 2 Zahlen kann also bei der Gleitkommadarstellung sehr klein aber auch sehr groß sein. Ziel ist hier eine konstante relative Genauigkeit zu erzielen, wogegen bei Integer Zahlen eine konstante absolute Genauigkeit erreicht wird.

f) Lösung: siehe Aufgabe 7 bei den Beobachtungen.

4) Zusatzaufgabe: Ermittlung von π nach Archimedes

Viele mathematische und naturwissenschaftliche Probleme können nicht oder nur schwer durch Angabe einer direkten Lösungsformel gelöst werden, stattdessen ist aber oftmals eine schrittweise Annäherung an die exakte Lösung möglich. Solche sogenannten iterativen Approximationen lassen sich meist algorithmisch beschreiben und in ein Computer-Programm umsetzen. Dabei ist allerdings große Sorgfalt geboten, wie die folgende Aufgabe zeigt.

Ein klassisches Beispiel für die iterative Approximation ist die Bestimmung der Kreiszahl π. Eines der ersten bekannten Verfahren geht auf Archimedes von Syrakus (um 250 v.Chr.) zurück. Die Formel Kreisumfang gleich zweimal Kreisradius mal π war Archimedes bereits bekannt. Durch immer genauere Approximation des Umfangs eines Kreises mit Radius eins mit Hilfe von in den Kreis einbeschriebenen Polygonen konnte er somit π approximieren. Für ihn war das eine mühselige Arbeit mit Tinte und Papyrus, wir können das heute dank Computer im Bruchteil einer Sekunde erledigen.

a) Berechnen Sie Seitenlänge und Umfang eines in den Einheitskreis einbeschriebenen Quadrats (vgl. Abbildung 2)!

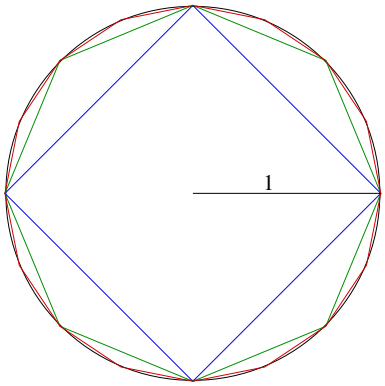


Abbildung 2: Einheitskreis und einbeschriebene Polygone

b) Durch Verdopplung der Ecken erhält man aus dem Quadrat sukzessive regelmäßige Achtecke, Sechszehnecke, etc. (vgl. Abbildung 3). Geben Sie eine Formel für die Seitenlänge s_{n+1} des 2^{n+1}-Ecks in Abhängigkeit von der Seitenlänge s_n des 2^n-Ecks an!

Hinweis: Fertigen Sie eine Skizze an und erinnern Sie sich an den Satz von Pythagoras!

c) Mit der in Teilaufgabe b) gefundenen iterativen Formel für die Seitenlänge der Polygone, kann man nun schrittweise Näherungen für π berechnen. Implementiert man die Formel zum Beispiel in einer `FOR`-Schleife, so ergeben sich aber folgende Werte:

Anzahl der Ecken	2^5	2^{10}	2^{20}	2^{25}	2^{30}
Approximation	$3,136548$	$3,141587$	$3,141596$	$3,142451$	$0,000000$

Welcher Teil der Formel aus Teilaufgabe b) ist für den auftretenden Fehler verantwortlich? Beheben Sie das Problem durch algebraische Umformungen!

Lösung:

a) Quadrat:

$$s = \sqrt{2}$$
$$U = 4\sqrt{2} \approx 5.6568$$

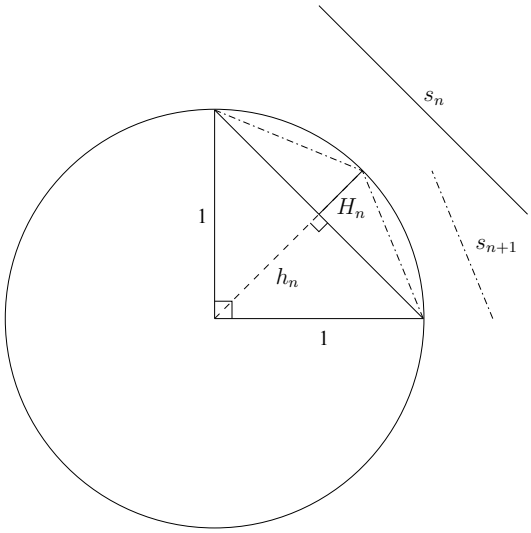


Abbildung 3: Definition der rekursiven Größen.

b) Es gilt mit den Definitionen aus Abbildung 3 (& Satz von Pythagoras!):

$$h_n^2 + \left(\frac{s_n}{2}\right)^2 = 1$$
$$H_n = 1 - h_n$$
$$H_n^2 + \left(\frac{s_n}{2}\right)^2 = s_{n+1}^2$$

und damit

$$\begin{aligned}
s_{n+1} &= \sqrt{H_n^2 + \left(\frac{s_n}{2}\right)^2} = \sqrt{(1-h_n)^2 + \left(\frac{s_n}{2}\right)^2} \\
&= \sqrt{\left(1 - \sqrt{1 - \left(\frac{s_n}{2}\right)^2}\right)^2 + \left(\frac{s_n}{2}\right)^2} \\
&= \sqrt{\left(1 - 2\sqrt{1 - \left(\frac{s_n}{2}\right)^2} + 1 - \left(\frac{s_n}{2}\right)^2\right) + \left(\frac{s_n}{2}\right)^2} \\
&= \sqrt{2 - 2\sqrt{1 - \left(\frac{s_n}{2}\right)^2}} \\
&= \sqrt{2 - \sqrt{4 - s_n^2}} \quad .
\end{aligned}$$

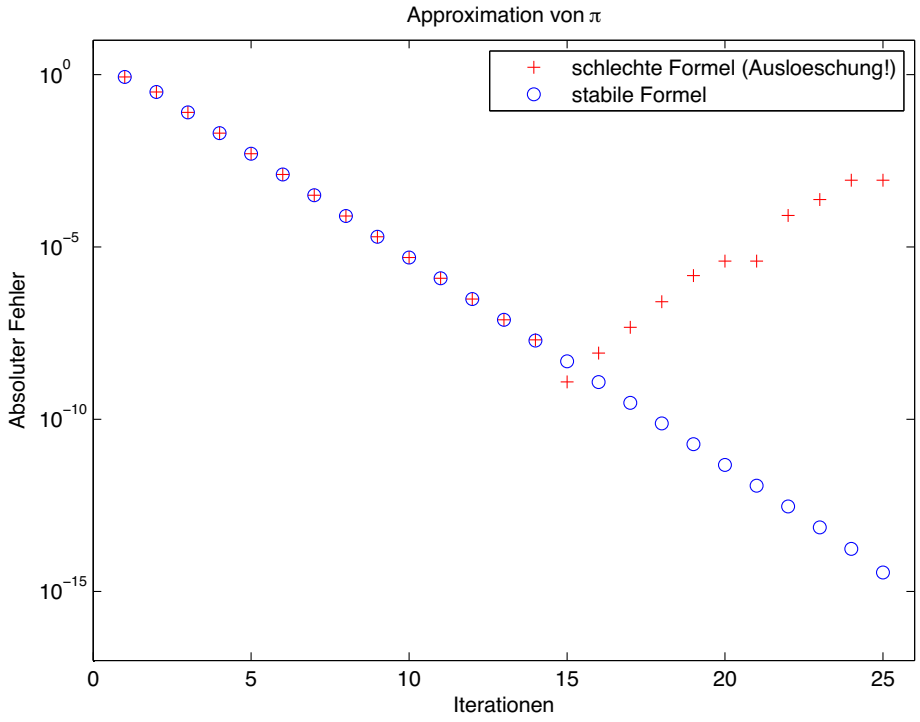
Der gesamte Umfang des 2^n-Ecks ergibt sich somit zu $U_n = 2^n \cdot s_n$ und damit also $\pi_n = U_n/2 = 2^{n-1} \cdot s_n$.

c) Effekt wie in Tabelle auf Angabe: Vermeintlich höhere Genauigkeit durch mehr Rekursionen verbessert das Ergebnis nicht, sondern zerstört den gesamten Wert!
Problem: Auslöschung!

Algebraische Umformung zur Vermeidung der Auslöschung:

$$\begin{aligned} s_{n+1} &= \sqrt{2 - \sqrt{4 - s_n^2}} \\ &= \sqrt{2 - \sqrt{4 - s_n^2}} \cdot \frac{\sqrt{2 + \sqrt{4 - s_n^2}}}{\sqrt{2 + \sqrt{4 - s_n^2}}} \\ &= \frac{|s_n|}{\sqrt{2 + \sqrt{4 - s_n^2}}} \end{aligned}$$

Vergleich der absoluten Fehler der beiden Formeln in Bezug auf echte Lösung π in semilogarithmischer Skala (erstellt mit matlab-Programm »archimedes.m« aus www):



5) Zusatzaufgabe: Fehlerabschätzung von Zeitschrittweiten

In Computerspielen werden für die Simulation von Physik oft Zeitschrittweiten Δt von $\frac{1}{50}$ oder $\frac{1}{60}$ gewählt um die Positionsänderung von einem Objekt innerhalb eines Frames zu simulieren.

Wir wollen hier den vereinfachten Fall mit konstanter Objekt-Geschwindigkeit und unter Vernachlässigung aller externen Kräfte und Kollisionen betrachten. Dann lässt sich die Positionsänderung eines Objektes innerhalb eines Zeitschrittes durch $\Delta x = \Delta t \cdot \vec{v}$ beschreiben, wobei Δx die Positionsänderung innerhalb eines Zeitschrittes ist und $\vec{v}$ die konstante Geschwindigkeit des Objektes.

Durch die vorangegangenen Aufgaben haben wir bereits festgestellt, dass Zahlen in Binärdarstellung nur eine begrenzte Genauigkeit besitzen, womit Δt evtl. nicht exakt gespeichert werden kann.

Geben Sie eine Abschätzung an, ob und wieviel Genauigkeit bei der Konvertierung der Zeitschrittweite $\frac{1}{60}$ zu 32-bit IEEE floating-point Zahlen verloren geht.

Inwiefern kann sich dieser Fehler auf sehr lange Simulationsläufe auswirken? Haben Sie eine Idee, dieses Problem zu umgehen bzw. den entstehenden Fehler zu klein zu halten?

Lösung:

Floating Point Darstellung von 1/60:

Wir starten mit einfacher Division:

$$
\begin{array}{rcccccccccccccc}
 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & : & 1 & 1 & 1 & 1 & 0 & 0 \\
-) & & 1 & 1 & 1 & 1 & 0 & 0 & & & & & & & \\
\hline
 & & & & & 1 & 0 & 0 & 0 & 0 & 0 & 0 & & & \\
-) & & & & & & 1 & 1 & 1 & 1 & 0 & 0 & & & \\
\hline
 & & & & & & & & & 1 & 0 & 0 & & &
\end{array}
\quad = \quad 0.000001\overline{0001} \text{ bzw. } 0.00\overline{0001}
$$

Für die Mantisse stehen 23 Bit zur Verfügung. Die erste »1« an der 6-ten Nachkommastelle wird allerdings nicht explizit abgespeichert. D. h. erst ab der 30-ten Nachkommastelle können die Ziffern nicht mehr abgespeichert werden:

	0,000001	0001	0001	0001	0001	0001	000[1
Nr. der Nachkommastelle:	7	11	15	19	23	27	

Mit der Normalisierung und dem nicht-Abspeichern der führenden »1« kann die Zahl deshalb bis auf die 29. Stelle genau abgespeichert werden.

Wie wird die Zahl dann als floating-point Zahl abgespeichert?
Sign: 0 für positive Zahlen
Exponent: $-6 + 127 = 0\text{b}1111001$
Mantisse: 0b00010001000100010001001 mit nach oben gerundeter letzten Stelle!

0	01111001	00010001000100010001001
Sign	Exponent	Mantissa

Floating point number: $(1) \cdot 2^{-6-23} \cdot 8947849 = 0.016666667535901069641113 2\ldots$

Kleines Beispielprogramm für Zweifler:

```
0   #include <iostream>

    int main()
    {
       float l = (1.0/60.0);
5      std::cout << l << std::endl;
       std::cout << (void*)*(unsigned long*)&l << std::endl;
    }
       /*
       Output:
10     0.0166667
       0x3c888889
       */
```

Wie kann man das Problem umgehen?

- Positionsänderung relativ zur Startposition berechnen.
- Als Zeitschrittweite eine 2er-Potenz wählen (z.B. $1/64$).
- Rechengenauigkeit erhöhen.
- Fehler ignorieren, wenn die Simulationsdauer klein genug ist.

6) Zusatzaufgabe: Ganzzahlen, Fest- und Gleitkommazahlen

In dieser Aufgabe schauen wir uns die Vorteile von Gleitkommazahlen gegenüber Festkommazahlen und den IEEE-Standard an. Binär Gleitkommazahlen haben das folgende format:

$$(-1)^{\text{Vorzeichen}} \cdot 2^{\text{Exponent}} \cdot \text{Mantisse}.$$

Wir vergleichen drei Arten in denen wir Zahlen mit 8 Bits speichern können:

- I: **Ganzzahlen** – 1 Bit für Vorzeichen (0 für »+«, 1 für »-«) und 7 Bits für den Ganzteil (in dieser Reihenfolge),
- F: **Festkommazahlen** – 1 Bit für Vorzeichen, 4 Bits für den Ganzteil und 3 Bits für die Nachkommastellen (in dieser Reihenfolge),
- G: **Gleitkommazahlen** – 1 Bit für Vorzeichen, 5 Bits für den Exponenten und 2 Bits für die Mantisse (in dieser Reihenfolge).
  In diese Aufgabe betrachten wir den Exponent als vorzeichenbehaftete Ganzzahl.
  Bei der Mantisse wird von einer Normalisierung mit führender Eins ausgegangen, die nicht gespeichert wird.

Aufgabe: Füllen Sie die fehlende Einträge der Tabelle aus:

Nr.	Frage		$A = I$	$A = F$	$A = G$
1	Darstellungs-beispiele	1 0000000	-0	$-0,0$	$-1 = -2^{+0} \cdot 2^{0}$
2		0 0000000	$+0$		
3		0 1000000	64		
4		0 1000100	68		
5		0 1000110	70	$8,75$	
6		0 1000111	71	$8,875$	$0,875 = 2^{-1} \cdot (2^{0} + 2^{-1} + 2^{-2})$

In Aufgaben Nr. 22–29 wird die Rundungsfunktion $\mathrm{rd}_{\mathrm{A}} : \mathbb{R} \to A$ gebraucht. Für jedes $A \subset \mathbb{R}$, ist sie definiert wie folgt:

$$\mathrm{rd}_{\mathrm{A}}(x) = a \in A, \text{ so dass } |x - a| \leq |x - b| \quad \forall b \in A \quad . \tag{1}$$

Zum Beispiel $\mathrm{rd}_{\mathrm{F}}(\pi) = 3,125$.

Nr.	Frage		$A = I$	$A = F$	$A = G$
7	Allg. Eigenschaften	$\lvert A\rvert$	255		
8		$\max_{a\in A} a$	127		
9		$\min_{a\in A} a$	-127		
10		$\min_{a\in A, a>0} a$	1		
11		Ist 0 in A?	ja		
12	Anzahl Zahlen von A in	$[2^{-15}, 2^{-14})$	0		
13		$[1, 2)$	1		
14		$[2, 4)$	2		
15		$[4, 8)$	4		
16		$[8, 16)$	8		
17		$[16, 32)$	16		
18		$[32, 64)$	32		
19		$[64, 128)$	64		
20		$[2^{15}, 2^{16})$	0		
21		$[0, 1)$	1		
22	Rundungen: $\mathrm{rd_A}(x)$	3^{-5}	0		2^{-8}
23		2,1	2		
24		3,1	3		
25		9	9		
26		18	18	15,875	
27		1023	127	15,875	
28	Absoluter Rundungsfehler: $\lvert x - \mathrm{rd_A}(x)\rvert$	3^{-5}	3^{-5}		
29		2,1	0,1		
30		3,1	0,1		
31		9	0		
32		18	0		
33		1023	...	...	
34	Relativer Rundungsfehler: $\left\lvert\frac{x-\mathrm{rd_A}(x)}{x}\right\rvert$	3^{-5}	1		
35		2,1	0,048		
36		3,1	0,032		
37		9	0		
38		18	0		
39		1023	...	...	

Beobachtungen

a) Betrachten Sie Ihre Antworten zu Fragen Nr. 2 und 3 für $A = G$. Was ist zu beobachten?

b) Betrachten Sie Ihre Antworten zu Fragen Nr. 25 und 26 für $A = G$. Was ist zu beobachten?

c) Betrachten Sie Ihre Antwort zu Frage Nr. 25 für $A = G$. In manche Programmiersprachen (z.B. JavaScript) gibt es kein Format für Ganzzahlen. Welche Konsequenzen kann das haben?

d) Was ist die kleinste Ganzzahl, die bei 32-Bit IEEE Gleitkommazahlen nicht exakt darstellbar ist?

e) Betrachten Sie die Antworten zu den Fragen 26.-27. für $A = F$. Welche Konsequenzen kann die Abwesenheit einer Inf-Darstellung haben?

f) Für eine Mantissendarstellung mit 2 Bits ist die Maschinengenauigkeit $\varepsilon_{\text{Ma}} = 2^{-3}$. Vergleichen Sie ihre Antworten zu Fragen Nr. 34-39 mit der Maschinengenauigkeit. Ist

$$\left|\frac{x - \text{rd}_{\text{G}}(x)}{x}\right| \leq \varepsilon_{\text{Ma}} \tag{2}$$

immer erfüllt?

g) Gibt es eine Relation zwischen der Maschinengenauigkeit und der Zahl aus Frage Nr. 10?

h) Was ist der Anteil an Zahlen im $[0, 1)$ beim Format G (ungefähr)? Bei 32-Bit IEEE?

i) Wie viele Zahlendarstellungen sind für spezielle Werte (Exponentenkombinationen $00\ldots0$ und $11\ldots1$) bei IEEE reserviert?

j) Gibt es einen Unterschied zwischen I und F? Betrachten Sie Ihre Antworten zu Fragen 12-21 für $A = I$ und $A = F$. Kann man F anhand I implementieren?

k) Die darstellbare Zahlen bei I und F sind **linear** verteilt. Wie sind die Zahlen G verteilt?

l) Was hängt von der Anzahl an Bits in der Mantisse ab?

m) Was hängt von der Anzahl an Bits in dem Exponent ab?

Das einfache Format G ist leider an mehrere Stellen mehrdeutig. Alle Mehrdeutigkeiten müssen in einem Standard wie dem IEEE-Standard gelöst werden um Konsistenz und Reproduzierbarkeit zu garantieren.

Lösung: Ausgefüllte Tabelle:

Nr.	Frage		$A = I$	$A = F$	$A = G$
1	Darstellungsbeispiele	1 0000000	-0	$-0,0$	$-2^{+0} \cdot 2^{0}$
2		0 0000000	$+0$	$+0,0$	$2^{+0} \cdot 2^{0}$
3		0 1000000	64	8,0	$2^{-0} \cdot 2^{0}$
4		0 1000100	68	8,5	$2^{-1} \cdot 2^{0}$
5		0 1000110	70	8,75	$0,75 = 2^{-1} \cdot (2^{0} + 2^{-1})$
6		0 1000111	71	8,875	$0,875 = 2^{-1} \cdot (2^{0} + 2^{-1} + 2^{-2})$
7	Allg. Eigenschaften	$\lvert A \rvert$	255	255	256
8		$\max_{a \in A} a$	127	15,875	$57344 = 2^{15} \cdot (2^{0} + 2^{-1} + 2^{-2})$
9		$\min_{a \in A} a$	-127	-15,875	-57344
10		$\min_{a \in A, a > 0} a$	1	0,125	$2^{-15} \cdot 2^{0}$
11		Ist 0 in A?	ja	ja	nein
12	Anzahl Zahlen von A im	$[2^{-15}, 2^{-14})$	0	0	4
13		$[1, 2)$	1	8	4
14		$[2, 4)$	2	16	4
15		$[4, 8)$	4	32	4
16		$[8, 16)$	8	64	4
17		$[16, 32)$	16	0	4
18		$[32, 64)$	32	0	4
19		$[64, 128)$	64	0	4
20		$[2^{15}, 2^{16})$	0	0	4
21		$[0, 1)$	1	8	$60 = (2^{4} - 1) \cdot 2^{2}$

Nr.	Frage		$A = I$	$A = F$	$A = G$
22	Rundungen: $\mathrm{rd}_A(x)$	3^{-5}	0	0	2^{-8}
23		2,1	2	2,125	2
24		3,1	3	3,125	3
25		9	9	9	8 (oder 10)
26		18	18	15,875	16 (oder 18)
27		1023	127	15,875	1024
28	Absoluter Rundungsfehler: $\lvert x - \mathrm{rd}_A(x)\rvert$	3^{-5}	3^{-5}	3^{-5}	$2{,}09{\cdot}10^{-4}$
29		2,1	0,1	0,025	0,1
30		3,1	0,1	0,025	0,1
31		9	0	0	1
32		18	0	2,125	2
33		1023	...	...	1
34	Relativer Rundungsfehler: $\left\lvert\frac{x-\mathrm{rd}_A(x)}{x}\right\rvert$	3^{-5}	1	1	0,051
35		2, 1	0,048	0,012	0,048
36		3, 1	0,032	0,008	0,032
37		9	0	0	0,111
38		18	0	0,118	0,111
39		1023	...	...	0,001

Beobachtungen

a) Betrachten Sie Ihre Antworten zu Fragen Nr. 2 und 3 für $A = G$. Was ist zu beobachten?

- Bei vorzeichenbehafteten Ganzzahlen ist 0 nicht eindeutig definiert.

b) Betrachten Sie Ihre Antworten zu Fragen Nr. 25 und 26 für $A = G$. Was ist zu beobachten?

- Ohne bestimmte Rundungsregeln gibt es Zahlen, die man nicht eindeutig runden kann.

c) Betrachten Sie Ihre Antwort zu Frage Nr. 25 für $A = G$. In manche Programmiersprachen (z.B. JavaScript) gibt es kein Format für Ganzzahlen. Welche Konsequenzen kann das haben?

- Es kann vorkommen, dass Rundungsfehlern auftreten, wenn man Objekte einfach zählen möchte.

d) Was ist die kleinste Ganzzahl, die bei 32-Bit IEEE Gleitkommazahlen nicht exakt darstellbar ist?

- $2^{24} + 1 \approx 17$ Millionen. Vergleichswerte: 1 Gigabyte $= 2^{30}$ Bytes,
  1 Mililiter Wasser enthält $\approx 2^{74}$ Moleküle.

e) Betrachten Sie die Antworten zu den Fragen 26.-27. für $A = F$. Welche Konsequenzen kann die Abwesenheit einer Inf-Darstellung haben?

- Man weiß nicht, ob 01111111_F genau 15,875 entspricht, oder etwas größerem.

f) Für eine Mantissendarstellung mit 2 Bits ist die Maschinengenauigkeit $\varepsilon_{\mathrm{Ma}} = 2^{-3}$. Vergleichen Sie ihre Antworten zu Fragen Nr. 34-39 mit der Maschinengenauigkeit. Ist

$$\left|\frac{x - \mathrm{rd}_{\mathrm{G}}(x)}{x}\right| \leq \varepsilon_{\mathrm{Ma}} \tag{3}$$

immer erfüllt?

- Die Rundungsfehler, die durch $\mathrm{rd_G}$ entstehen sind immer kleiner oder gleich $\varepsilon_{\mathrm{Ma}}$ (solange die Zahl im Range ist, natürlich). Man kann Gleichung (3) beweisen (betrachten Sie dazu Ihre Antworten zu Fragen Nr. 12-20.).

g) Gibt es eine Relation zwischen der Maschinengenauigkeit und der Zahl aus Frage Nr. 10?

- Nein. Die kleinste darstellbare positive Zahl ist nur von der Anzahl an Bits in den Exponenten abhängig. Die Maschinengenauigkeit ist nur von der Anzahl an Bits in der Mantisse abhängig.

h) Was ist der Anteil an Zahlen im $[0, 1)$ beim Format G (ungefähr)? Bei 32-Bit IEEE?

- Fast ein Viertel für beide. Bei G: $\frac{60}{256}$ =0,23.

i) Wie viele Zahlendarstellungen sind für spezielle Werte (Exponentenkombinationen $00\ldots0$ $11\ldots1$) bei IEEE reserviert?

- 2 (Vorzeichen) $\cdot$ 2 (Exponentenkombinationen) $\cdot$ 2^{23} (Mantissenmöglichkeiten) = $2^{25} = 33554432$ Zahlendarstellungen.

j) Gibt es einen Unterschied zwischen I und F? Betrachten Sie Ihre Antworten zu Fragen 12-21 für $A = I$ und $A = F$. Kann man F anhand I implementieren?

- Es gibt kaum Unterschiede. Man kann F durch I und einen Skalierungsfaktor implementieren.

k) Die darstellbare Zahlen bei I und F sind **linear** verteilt. Wie sind die Zahlen G verteilt?

- Logarithmisch.

l) Was hängt von der Anzahl an Bits in der Mantisse ab?

- Die Anzahl an darstellbare Zahlen im $[2^k, 2^{k+1})$ und die Maschinengenauigkeit ε_{Ma}.

m) Was hängt von der Anzahl an Bits in dem Exponent ab?

- Die Zahlen aus Fragen Nr. 8-10.