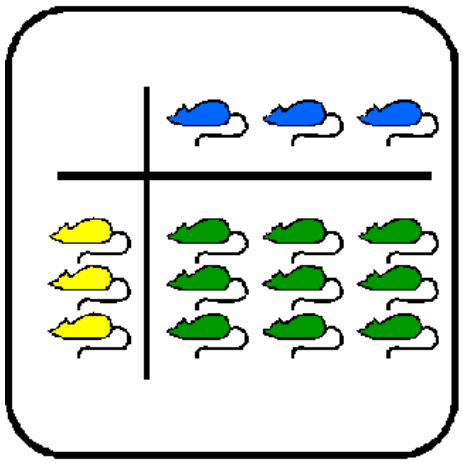
Multiplikation



Rundungsfehler

Beitrag zu "Werkstattunterricht Multiplikation"

Allgemeine Didaktik - Seminar SS95

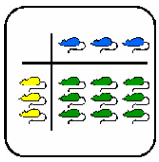
Autor:

Ernesto Ruggiano
Oberwiesenstr. 42
8050 Zürich

eruggian@inf.ethz.ch

Betreuer

Prof. Werner Hartmann



Thema:	endliche Arithmetik (rechnen mit einem Computer)
Schultyp:	Gymnasium (die letzten zwei Jahre); Fachhochschule; technische Berufsschule
Vorkenntnisse:	elementare Programmierkenntnisse; mathematische Umformungen; binäre Darstellung der Zahlen
Bearbeitungsdauer:	90 min.
Fassung vom:	14.7.95
Schulerprobung:	nein

Übersicht

Rechnen im mathematischen Sinn und Rechnen im Bereich der Informatik ist nicht dasselbe. Für das eine gilt die saubere mathematische Abstraktion, für das andere die nicht immer einfache praktische Anwendung. Die klassischen Probleme, die daraus hervorgehen, werden in diesem Posten behandelt. Die Schüler finden zuerst eine Definition für die sogenannten Maschinenzahlen. Anhand dieser Zahlen werden dann die Probleme illustriert und erklärt.

Bemerkung: Dieser Posten könnte so aufgebaut werden, dass er in direkter Verbindung mit dem Posten über Zufallsgeneratoren steht. Der Einführungsteil des Auftrags könnte sich explizit auf jenen Posten beziehen. Ich denke, dass dieser Posten von den Schülern direkt anschliessend an jenen über Zufallsgeneratoren behandelt werden kann. Sie bekommen damit sofort eine Ahnung von den praktischen Problemen, mit denen man konfrontiert wird, wenn man mit Computern rechnet. Ausserdem haben sie die Möglichkeit zu beobachten, wie verschiedene Aspekte der Informatik miteinander vernetzt sind. Diese Erfahrung kann ihnen sehr nützlich sein, wenn sie sich mit anderen Gebieten auseinandersetzen werden: die verschiedenen Gebiete stehen nicht für sich allein da, sondern immer in Beziehung zueinander!

Lernziele

Das Hauptziel, eigentlich die Motivation für diesen Posten, ist (wie schon für den Posten Zufallsgeneratoren) den Schülern zu zeigen, wie viele Einsichten hinter anscheinend so trivialen mathematischen Operationen wie Multiplikation, Subtraktion oder Potenzieren stecken.

Als konkrete Ziele nimmt sich dieser Posten folgendes vor:

- Die Schüler kennen die Definition von *Maschinenzahl*; *Rundungsfehler*; *Maschinengenauigkeit* und *numerische Auslöschung*.
- Sie wissen, dass sie, für einen Programmtest Werte die an den Grenzen liegen (d.h. sehr grosse oder sehr kleine Werte) benützen müssen.
- Sie haben eine Ahnung von der Lösung einiger konkreter Probleme (nur für jene, die Teil III der Theorie gelesen haben)

Material

- Theorie: Rechnen mit Computern
- Der Computer und die im Unterricht normalerweise benützte Programmiersprache (für Auftrag 2)

Quellen

[1] W.Gander: *Computer Mathematik* , 1992 Birkhäuser , 2te Auflag
[2] R.Sedgewick: *Algorithms* , 1988 Addison Wesley

Hinweise, Lösungen

Hinweis:

Ich habe zwei konkrete Behandlungen von Overflowsfälle geschrieben. Ich stelle es hier für interessierten Lehrer und Studenten zu Verfügung.

Teil III: Overflow Behandlung

(freiwillig Auftrag)

Ein Overflow passiert, normalerweise, bei der Multiplikation von zwei Elementen von [-M, M], deren Resultat aus dem Rechenbereich herausfällt. In vielen Fällen kann man jedoch etwas tun, um dies zu verhindern. Untersuchen wir die folgenden Beispiele:

1) r = (r * a + c) mod m (vgl. Werkstattposten *Zufallsgeneratoren*)
2) $r = x^2 + y^2$

1) Nehmen wir die folgenden Werte als gegeben:

a) Der Computer stellt die Zahlen mit einer Sequenz von 32-bit dar. Die grösste darstellbare Zahl M ist demzufolge: $2^{32} = 4, 29 \cdot 10^9$ (vgl. die obige binäre Darstellung).
b) $m = 10^8$, c = 1, $a = 1, 234567 \cdot 10^6$, $b = r_o = 3, 1415821 \cdot 10^7$.

Alle diese Werte sind kleiner als M. Im ersten Iterationsschritt rechnen wir aber a * b, dessen Ergebnis in der Grössenordnung von 10^{13} und demzufolge grösser als M ist. Es ist festzuhalten, dass unser Endergebnis r kleiner als M ist, weil es mit modulo 10^8 berechnet wird (d.h. es ist eine Zahl mit höchstens 8 Ziffern).
Schlussfolgerung: obwohl uns nur die letzten 8 Ziffern der Multiplikation a * b interessieren, kann r nicht ausgerechnet werden, weil diese ein Zwischenresultat liefert, das kein Element des Bereichs [-M, M] ist.
Wir können aber das Problem umgehen, indem wir die Multiplikation zerlegen.
Wir definieren $a_1 = a \; div \, 10^4$, $a_0 = a \bmod 10^4$ und analog $b_1 = b \; div \, 10^4$, $b_0 = b \bmod 10^4$.
Dann haben wir:

Der Term (1) ist bestimmt kleiner als $(10^4)^2 = 10^8$ (dies folgt gerade aus der Definition von a_0 und b_0). Demzufolge wird er eine Zahl mit 8 Ziffern sein: (1) = $D_7 \ldots D_0$.
Vom zweiten Term (2) = $D_9 \ldots D_0$ interessieren uns nur $D_3 \ldots D_0$ (d. h. die ersten vier Ziffern), da er mit 10^4 multipliziert wird.(10^4*(2) = $D_9 \ldots D_0 0000$). Diese Ziffern werden folgendermassen berechnet: ((2)mod10^4)*10^4, da (2)mod10^4 = $0 \ldots 0 D_3 D_2 D_1 D_0$.
Den dritten Term brauchen wir nicht, da er mit 10^8 multipliziert wird (beachte: genau dieser Faktor verursacht den Overflow-error).
Demzufolge können wir die ersten 8 Ziffern der Multiplikation a*b ausrechnen:

$$mult_8 = ((a_0 b_1 + b_0 a_1) \bmod 10^4) \cdot 10^4 + a_0 b_0) \bmod m$$

Zuletzt gelingt es uns r_1 = (a*b + c) mod m auszurechnen, indem wir a*b durch $mult_8$ ersetzen:

$$r_1 = (mult_8 + c) \bmod m$$

2) Nehmen wir M = 3, 4 . 10^{38} . Es besteht die Gefahr, das mit zu grossen x und y, x^2 bzw. y^2 aus dem Bereich (-M, M) herausfallen. Wie können wir das Problem lösen?
Analog zum Beispiel **1)** suchen wir nach einer Umformung der Gleichung $r = \sqrt{x^2 + y^2}$, um das Risiko eines Overflow zu vermeiden. Wir können die Formel mit einem kostanten Wert multiplizieren:

$$r = \sqrt{x^2 + y^2} = \frac{m}{m}\sqrt{x^2 + y^2} = m\sqrt{\frac{x^2 + y^2}{m^2}}$$

Wenn m gleich das grösste Zahl zwischen x und y (nehmen wir x an) ist, erhalten wir.:

$$r = x\sqrt{\frac{x^2 + y^2}{x^2}} = x\sqrt{\frac{x^2}{x^2} + \frac{y^2}{x^2}} = x\sqrt{1 + \left(\frac{y}{x}\right)^2}$$

Also es gilt $r = x\sqrt{1+z}$ wo z < 1! Die Gefahr eines Overflow ist so gebannt.

Lösung Auftrag 2

Für die interessierten Lehrer kommt die Formel um π zu berechnen von folgendem Grenzwert her:

$$\lim_{n \to \infty} \frac{A_n}{r^2} = \pi$$

wo A_n die Fläche des im Kreis eingeschriebenen Vielecks ist.
Für r =1 gilt:

$$A_6 = \frac{3}{2}\sqrt{3} \quad \text{und} \quad A_n = \frac{n}{2}\sin\alpha_n \text{ , wobei } \quad \alpha_n = \frac{2\pi}{n}$$

Indem man die trigonometrische Formel benutzt:

$$\sin\left(\frac{\alpha}{2}\right) = \sqrt{\frac{1-\cos\alpha}{2}} = \sqrt{\frac{1-\sqrt{1-\sin\alpha}}{2}}$$

erhalten wir einen rekursive Formel um, von A_n ausgehend, A_{2n} zu berechnen (vgl. [1]).

Das Problem des Programms des Auftrages besteht in der Subtraktion $1\text{-}s^2$ in der zweiten Wurzel:
Da s immer kleiner wird, nähert sich $\sqrt{1-s^2}$ immer mehr 1 an, so dass:

$$\sqrt{\frac{1 \simeq 1}{2}} \cong 0 \quad \text{(numerische Auslöschung!)}$$

Um den Fehler zu vermeiden, kann man die folgende Identität benützen:

$$a - b = \frac{a^2 - b^2}{a + b}$$

wobei a=1 und b=$\sqrt{1 - s^2}$.
Auf diese Weise erhalten wir folgenden Algorithmus (aus [1]) :

```
program pistabil;
var a, aneu,s: real; n:Longint;
begin
   s := sqrt(3)/2; aneu := 3*s; n:=6;
   repeat
      a:= aneu;
      s:= s/sqrt(2*(1+sqrt((1+s)*(1-s))));
      n:= 2*n; aneu := n/2*s;
      writeln(n :8,aneu : 16 : 10,s : 20)
   until aneu<=a;
   readln
end.
```

Dieser Algorithmus liefert uns die folgenden Ergebnisse (aus [1]):

n	An	sin α_n
12	3.0	5.0E-01
24	3.1058285412	2.588190451E-01
48	3.1326286133	1.305261922E-01
96	3.1393502030	6.540312923E-02
192	3.1410319509	3.271908282E-02
384	3.1414524723	1.636173162E-02
768	3.1415576079	8.181139603E-03
1536	3.1415838921	4.090604026E-03
3072	3.1415904632	2.045306291E-03
6144	3.1415921060	1.022653680E-03
12288	3.1415925167	5.113269070E-04
24576	3.1415926194	2.556634618E-04
49152	3.1415926450	1.278317319E-04
98304	3.1415926514	6.391586611E-05
196608	3.1415926530	3.195793307E-05
393216	3.1415926534	1.597896654E-05
786432	3.1415926535	7.989483270E-06
1572864	3.1415926535	3.994741635E-06
3145728	3.1415926536	1.997370817E-06

Bemerkung: In diesem Algorithmus wurde im Vergleich zu jenem des Beginns die Bedingung der **While**-Schleife verändert (durch ein repeat ersetzt). Da die neue Formel "monoton wachsend" konvergiert, können wir die Iteration unterbrechen wenn $A_{2n} \leq A_n$ gilt.

Das Programm wird somit sogar maschinenunabhängig (d.h. wir müssen die Maschinengenauigkeit nicht mehr benützen!).

Lehrer-LernKontrolle / Test

Aufgabe 1

In der Theorie haben wir die Darstellung der reellen Zahlen durch den Computer genauer betrachtet:
Eine Zahl [ ] wird durch die Formel

$$\tilde{a} \in IM : \tilde{a} = \pm m.10^{\pm e}$$

dargestellt, wo **m** Mantisse genannt wird und **e** Exponent.

Betrachten wir folgenden Fall von Maschinenzahlen:

$m = D_0, D_1$ wobei $D_0, D_1 \in [0 \ldots 9]$
$e = D_2$ wobei $D_2 \in [0 \ldots 9]$

Zum Beispiel gehören $a = 7,5 \cdot 10^4$ und [ ] zu unserer Darstellung.
a)Wie viele Maschinenzahlen mit $D_0 \neq 0$ sind in dieser Darstellung enthalten?
b)Welches ist die grösste darstellbare Zahl (M)? (theoretischer Wert)
c)Welches ist die kleinste darstellbare Zahl (m)? (theoretischer Wert)
d)Welche Zahl ist die Maschinengenauigkeit ε?

Die Antworten sollen auch die dazu führenden Berechnungen, falls diese nötig sind, enthalten.
Für eine befriedigende Antwort genügt es, 2-3 Fragen zu beantworten.

Aufgabe 2:

In der Theorie haben wir die Zahl □ (epsilon), auch Maschinengenauigkeit genannt, definiert. Jetzt stellt man sich die Frage: welchen Wert besitzt diese Zahl bei deinen Computer?
Schreibe ein kleines Programm, das die Zahl □ berechnet. Erkläre dann dein Programm mit 3-4 Sätzen als Kommentar. Deine Antwort kann befriedigend auch ohne das Programm zu schreiben. In diesem Fall beschreibe in etwa 10 Sätzen, welche Methode du anwenden würdest, um □ zu berechnen.

Hinweis: Überlege: wie wird □ definiert?

Aufgabe 3 (freiwillig)

In der Theorie haben wir M als grösste darstellbare positive Maschinenzahl definiert. Wie ist es deiner Meinung nach möglich, zu wissen wie gross M für einen bestimmten Computer ist? Kann man M berechnen, ohne einen Overflow zu riskieren?
Kommentiere deine Überlegungen mit einigen Sätzen. Die Bewertung ist befriedigend, wenn deine Ideen plausibel sind.

Lösungen und Taxierung

Aufgabe 1

a) gegeben:

- Anzahl verschiedener Exponenten = 19 [-9, -8, ..., 9]
- Anzahl verschiedener D_0 (bei Mantisse) = 9 [1 ... 9] ($D_0 \neq 0$!)
- Anzahl verschiedener $D_1 = 10$

Daraus folgt:

- ☐ Anzahl positiver Zahlen: 19 * 10 *9 = 1710
- ☐ Wir dürfen die negativen Zahlen nicht vergessen, also:
- ☐ Totalanzahl Zahlen: 1710 * **2** = 3420

Wir sind aber noch nicht ganz fertig, denn die Zahl Null = $0,0.10^0$ wurde nicht berücksichtigt.

- ☐ Insgesamt haben wir also **3421** Zahlen.

b) $M = 9,9 \cdot 10^9$
c) $m = 1,0 \cdot 10^{-9}$
d) Wir rechnen:

$$1,0 \cdot 10^0 + 5 \cdot 10^{-2} = 1,05 \cong 1,1 \cdot 10^0$$

$$1,0 \cdot 10^0 + 4 \cdot 10^{-2} = 1,04 \cong 1,0 \cdot 10^0$$

$$\text{also: } \square = 5 \cdot 10^{-2}$$

Die Fragen b) und c) erfordern einen einfache Wiedergabe des in der Theorie Gelernten (vgl. Theorie nach **Definition 2.** Sie gehören demzufolge zum Niveau **K1**.
Die Punkte a) und d) erfordern hingegen einige Berechnungen. Ausserdem gibt es in der Theorie keine ähnlichen Probleme wie a) und d). Die Fragestellungen sind deshalb für die Schüler neu. Um diese beiden Punkte zu lösen, muss der Schüler das eben Gelernte in die Praxis umsetzen. Ich stufe diese beiden Probleme als Niveau **K3** (Anwendung) ein, auch wenn ich finde, dass die Lösung von Punkt d) weniger offensichtlich als jene von Punkt a) ist.

Aufgabe2

```
program Maschinengenauigkeit;
var eps;
begin
   eps := 1;
   while 1+ eps ≠ 1 do
   eps ;= eps/2;
   end;
   writeln("eps= ", 2*eps);
end.
```

Um eine Lösung zu erhalten, braucht der Schüler zwei Denkprozesse:
1) Er muss die Definition von □ kennen (um eine korrekte While-Schleife Bedingung zu schreiben).
2) Er muss eine Ahnung davon haben, wie man eine Zahl immer mehr verkleinern kann. Einige könnten z. B. auf die Idee kommen, statt der Division die Substrahierung zu benützen, z. B. eps = eps - 0, 1 , was zu einem sehr wenig leistungsfähigen Programm führt.
Auch diese Aufgabe scheint mir deshalb zum Typus **K3** zu gehören: der Schüler muss das eben Gelernte anwenden und ausserdem eine Idee haben, um etwas Neues zu lösen (wie in einer Lernaufgabe!).

Aufgabe3

Es ist tatsächlich unmöglich, M zu berechnen, ohne einen Overflowerror zu erhalten: um zu erfahren, welches die grösste Zahl ist, müssen wir unbedingt die Grenzen des Computers überschreiten. Ansonsten können wir nicht sicher sein, dass die berechnete Zahl tatsächlich die grösste ist! Wir müssen M bei jede Iteration auf dem Bildschirm schreiben, so dass wir die letzte berechnete Wert (vor den Absturz) lesen können.
Man kann hingegen M mit der Beziehung : M = 1/m runden. Nicht immer aber sind die Bereiche des Computers symmetrisch (d. h. M = 1 / (α *m)).

Die Frage erfordert einige Überlegungen. Der Schüler muss mehrere Begriffe zueinander in Beziehung setzen. Um seine Ideen zu argumentieren benötigt er eine Synthese. Die Aufgabe gehört deshalb zum Typus **K5**.

Werkstatt Multiplikation Auftragsblatt
Posten: **Rundungsfehler**

Was soll ich hier tun?

Betrachten wir eine Multiplikation :

a*b = 1234567890987654321 * 98765432101234567890 = ?

Kann diese Rechnung mit einem Computer gemacht werden ?
Es besteht das Risiko, dass **a*b** als Ergebnis eine sehr grosse Zahl liefert, und dass der Computer diese nicht mehr zu berechnen vermag. Im besten Fall meldet er dies mit einem sogenannten "**Overflowerror**", im schlimmsten Fall stürzt er ab.
In diesem Posten werden wie erklären, was diese Overflowerrors sind, und werden andere mögliche Fehler betrachten, die beim Rechnen mit dem Computer auftreten können.

Für diesen Posten ist folgender Ablauf vorgesehen:

(1) Studiere **Teil I** und **Teil II** der Theorie allein: *"Rechnen mit einem Computer"*.
45 Minuten sollten dazu genügen.

(2) Berechne mit dem Computer Π (siehe nächste Auftragsblatt)
45 Minuten.

Werkstatt Multiplikation Auftragsblatt
Posten:**Rundungsfehler**

Berechnung von Π durch dem Computer

Der reine Mathematiker beweist, dass es eine π genannte Zahl gibt, die durch die Formel

$$A(r) = \pi r^2$$

für die Kreisfläche definiert ist..
Der angewandte Mathematiker muss eine Nährungslösung für diese reelle Zahl finden.
Der reine Mathematiker, Mister *MM*, weiss, dass man um π zu berechnen, folgende rekursive Formel benützen kann:

$$A_6 = \frac{3}{2}\sqrt{3} \quad \text{und}$$

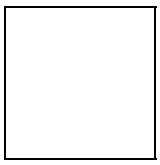
$$A_{2n} = \frac{n}{2}\sqrt{\frac{1-\sqrt{1-A_n^2}}{2}}$$

Er entwickelt deshalb folgenden Algorithmus:

```
program pi;
var a,s:real; n:integer;
begin
   s:= sqrt(3)/2; a:= 3*s; n:=6;
   while s>1E-5 do
   begin
      s:= sqrt((1-sqrt(1-s*s))/2);
      n:=2*n; a:= n/2*s;
      writeln(n:8,a:10:5,s:11);
   end;
   readln
end.
```

Während der Ausführung bemerkt er aber, dass ihm sein Algorithmus seltsame Ergebnisse liefert:

n	An
12	3.0000000000
24	3.1058285412
48	3.1326286133
96	3.1393502029
192	3.1410319508
384	3.1414524712
768	3.1415575977
1536	3.1415838514
3072	3.1415904255
6144	3.1415916208
12288	3.1415895717
24576	3.1415868397
-16384	-1.0471956132
-323768	-1.0471664706
0	0.0000000000
0	0.0000000000



Werkstatt Multiplikation Auftragsblatt
Posten:**Rundungsfehler**

Am Anfang läuft  in die richtige Richtung, doch dann wird es plötzlich negativ und dann sogar gleich Null!
Es hat seinen Algorithmus wieder und wieder durchgesehen, aber er findet einfach den Fehler nicht. Dies kommt daher, dass er nicht daran gedacht hat, dass seine Formel, wenn sie mit kleinen Werten von s gebracht wird, Fehlern unterworfen ist.
Kannst du Mister MM helfen, indem du den Fehler in seiner Formel findest?
Kannst du den Fehler korrigieren und einen funktionierenden Algorithmus, der π berechnet, für Mister MM entwerfen kannst?
Für die erste Antwort genügen ein paar Sätze, in denen erklärt wird, wo und weshalb die Formel einem Fehler unterworfen ist.
Für die zweite Antwort musst du versuchen:
1) die Formel mit Papier und Bleistift so umzuformen, dass sie keinem Fehler mehr unterworfen ist
2) einen kleinen Algorithmus zu entwerfen, der π nach der neuen Formel berechnet.

Es ist nicht immer einfach eine neue Formel zu finden, die nicht ebenfalls Fehlern unterworfen ist. Für eine befriedigende Lösung dieser Aufgabe genügt es deshalb, die erste Frage korrekt zu beantworten, und die zweite zu versuchen.
Für diese Aufgabe könnt ihr paarweise oder in kleinen Gruppen arbeiten. Wenn ihr in Gruppen arbeitet, diskutiert doch die neue Formel, bevor ihr sie programmiert.

Zeit: 45 Minuten

Rechnungen mit dem Computer

Der Rechner, wie das Wort schon selbst aussagt, rechnet, das heisst er führt Operationen aus. Mit dem Computer kann man z.B. die vier Basisoperationen der Arithmetik (+,-, x,/) ausführen. Wir wollen uns nun die Grenzen und die Probleme, die beim Rechnen mit Computern auftreten können, ein wenig näher anschauen.

Teil I: Maschinenzahlen

Zuallererst müssen wir uns bewusst werden, dass zwischen dem abstrakten Begriff von Zahl in der Mathematik und der Darstellung einer Zahl im Computer ein Unterschied besteht. Eine Zahl wird im Computer durch eine Serie von binären Zahlen dargestellt, die von links nach rechts gelesen werden, z.B. :

$11010010 = 1.2^7 + 1.2^6 + 0.2^5 + 1.2^4 + 0.2^3 + 0.2^2 + 1.2^1 + 0.2^0 = 210$

2^7	2^6	□	2^4	□	2^2	2^1	2^0
1	**1**	**0**	**1**	**0**	**0**	**1**	**0**
$1*2^7$	$1*2^6$	0*□	$1*2^4$	0*□	$0*2^2$	$1*2^1$	$0*2^0$

Wie wir wissen, kann eine Zahl in der Mathematik zu *IN*, zu ~~*Z*~~, zu *Q*, zu *IR* oder zu *C* gehören.
Wenn unsere Zahl zu *IN* gehört, gibt es bei der Darstellung mit dem Computer keine Probleme: es genügt, sie in eine binäre Zahl umzuwandeln (s. obiges Beispiel).
Wenn sie zu ~~*Z*~~ gehört, wird man ein Bit für die Zeichen + oder - anfügen müssen, aber auch dies ist kein grosses Problem.
Wenn sie hingegen zu *IR* gehört (die komplexen Zahlen ziehen wir hier nicht in Betracht) haben wir es mit einem Problem zu tun: wie sollen wir eine unendliche Serie von Zahlen darstellen? Wie sollen wir zum Beispiel π = 3, 141592653 ...auf dem Computer darstellen?
Dies ist natürlich unmöglich, da der Computer einen **endlichen** Speicher hat. Wir müssen uns deshalb damit begnügen, eine Rundung des reellen Wertes von π oder einer anderen reellen Zahl darzustellen. Wir können sagen, dass in einem Computer π korrekt bis zur zehnten Zahl nach dem Komma dargestellt wird.
Diese Nährungswerte stellen die Menge der Zahlen dar, mit denen der Computer arbeitet.

Definition 1:
Die Menge der Zahlen, mit denen der Computer arbeitet, heisst **Maschinenzahlmenge.** Als Symbol benützen wir *IM*.

Jede Zahl von *IM* stellt unendlich viele Zahlen von *IR* dar:
zum Beispiel stellt x = 1, 73215469, x □ *IM*, alle jene reellen Zahlen zwischen 1 und 2 dar, die die ersten acht gleichen Zahlen nach dem Komma wie x besitzen.
So stellt x die folgenden Zahlen dar:

1, 732154691...
1, 732154692 ...
1, 7321546931 ...
1, 73215469328173 ...

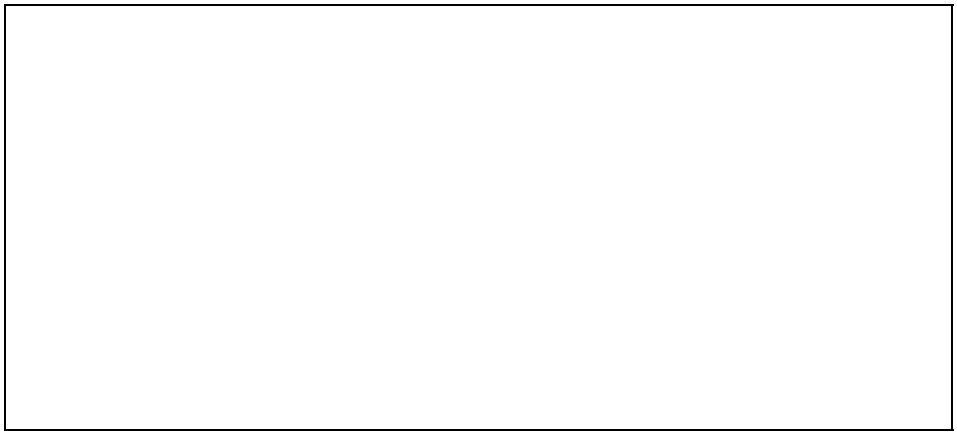


Bild 1: Bereich eines Maschinenzahls

Alle reellen Zahlen im Intervall $[x_a...x_b]$ werden im Computer durch die Zahl x_i gerundet.

Definition 2:
Jede Zahl $\tilde{a}$, die zu *IM* gehört, wird **maschinelle** Zahl genannt. Sie wird wie folgt dargestellt:

$$\tilde{a} \in IM: \quad \tilde{a} = \pm m.10^{\pm e}$$

wo

- m = D, D ... D **Mantisse** genannt wird
- e = D ..... D **Exponent** genannt wird
- D = eine Ziffer $\in$ [0, ..., 9]

Die Anzahl der Ziffern der Mantisse und des Exponenten hängt von der Anzahl der Bit ab, die zur Darstellung der Zahl gebraucht werden (normalerweise benützt man eine Potenz von 2, d.h. 8, 16, 32, 64, ...). Manchmal kann diese Anzahl aber auch von der benützten Programmierungssprache abhängen.
Da das Computer nur endlich viele Zahlen darstellen kann, existieren eine **grösste Maschinenzahl M** und eine **kleinste Maschinenzahl -M**:

$M = 9, 9 ... 9 \times 10^{+9 ... 9}$

$-M = 9, 9 ... 9 \times 10^{-9 ... 9}$

(theoretische erreichbaren Werten)

Bemerkung**:** In der Regel ist die erste Ziffer der Mantisse (jene links des Kommas) verschieden von Null.

Der **Rechenbereich** eines Computers ist demzufolge **[-M, M]**. Die reellen Zahlen die entweder grösser als M oder kleiner als -M sind, sind nicht darstellbar, da sie im sogenannten **Überlaufbereich** (overflow) hängen, d.h. der Computer meldet einen *Overflow error* . Als Beispiel :

M der Macintosh ist gleich $3,4 \cdot 10^{38}$. und
M der Sun ist gleich $1,0 \cdot 10^{308}$!

Im letzten Teil der Theorie (Teil III) werdet ihr einige Beispiele zur Verhinderung von potentiellen *Overflow errors* finden.

So wie es eine grösste positive Maschinenzahl gibt, existiert auch eine **kleinste positive** Maschinenzahl **m**:

$$m = 1,0...0 * 10^{-99..9}$$

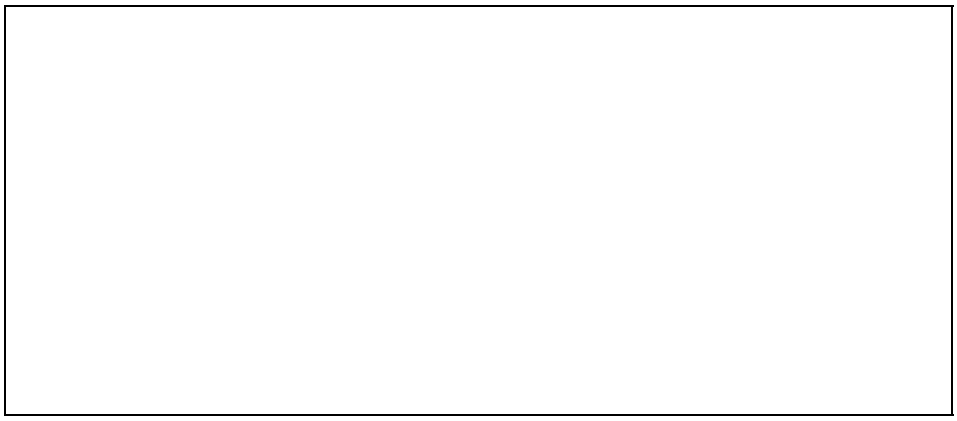


Bild 2: kleinste positive Maschinenezahl

m ist die erste, von Null verschiedene, positive Zahl ($\in$ *IM)*. Das bedeutet: Rechnungen, die m reduzieren (wie m/2 oder m^2), haben als Resultat Null, weil der Computer keine kleinere positive Zahl als m kennt und deshalb das Resultat auf Null rundet. Die Bereiche (-m, 0) und (0, m) heissen **Unterlaufbereiche** (*Underflow*). In diese Bereiche hineinzugeraten kann gefährlich sein, weil der Computer dann ständig nur mit Null rechnet!
Als Beispiel nehmen wir an:

$$m = 1,75 \cdot 10^{-38}$$

(Das ist die kleinste pos. Maschinenzahl des Macintosh, jene der Sun ist $m = 4,9407 \cdot 10^{-324}$)
Mit der Kreisgleichung $r = \sqrt{x^2 + y^2}$ und mit [ ], $y = 10^{-21}$ wird r gleich Null sein!
Eine eventuell darauffolgende Teilung durch r wird also einen Fehler in der Ausführung des Programms zur Folge haben.

Diese Fehler sind, wenn das Programm einmal gemacht ist, schwer zu finden (auf englisch heissen sie *bugs*). Dies hängt von der Tatsache ab, dass sie nur bei Grenzfällen auftreten, wie im obigen Beispiel, wo sowohl x als auch y sehr klein sind. Aufgabe des Programmierers ist es deshalb auch, das Programm auf mögliche *Underflow errors* zu testen, und sie eventuell zu korrigieren.

Teil II: Rundungsfehler

Wie wir in Teil I gesehen haben, stellt jede Maschinenzahl unendlich viele reelle Zahlen dar, deshalb sagt man, dass jede Maschinenzahl unendlich viele Zahlen, die Elemente von *IR* sind, **rundet.**
Es ist deshalb klar, dass eine Operation (auch eine einfache wie +, -, x, /) mit Zahlen, die zu *IM* gehören, ein Ergebnis geben wird, das ebenfalls Element von *IM* ist. Die Aufgabe der Informatiker (im Bereich der sogenannten numerischen Mathematik) besteht darin, sich zu vergewissern, dass das Maschinenresultat noch das reelle Ergebnis darstellt:

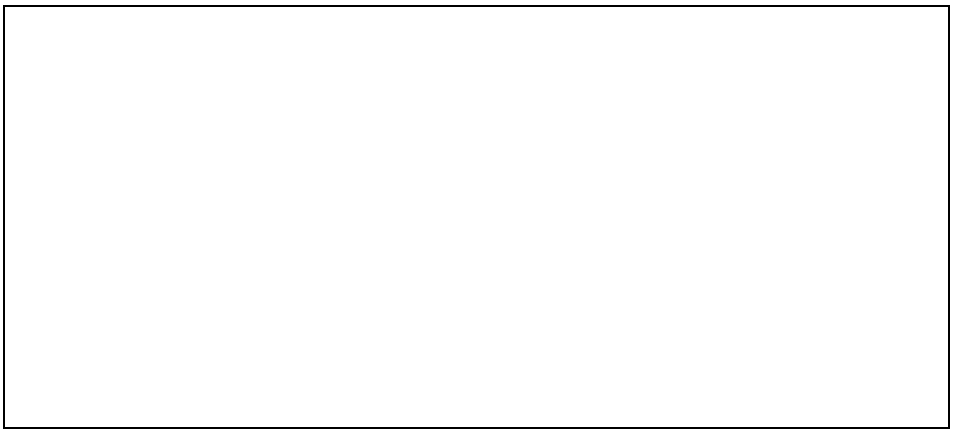


Bild 3: Maschinenresultat

a,b und r sind die Operanden bzw. das Resultat in *IR*
$\tilde{a}, \tilde{b}$ und $\tilde{r}$ sind die Operanden bzw. das Resultat in *IM*

Ein Beispiel: wir möchten das Produkt zwei Maschinenzahlen mit einem 8-bit Computer auszuführen:

$$\tilde{a} = 1{,}2345678 \text{ , } \tilde{b} = 1{,}1111111.$$

Da die Operanden 8-stellig sind ist das Product 16-stellig : $\tilde{a} * \tilde{b}$ = r = 1,37174198628258
aber das Computer besitzt nur 8 Stelle nach das Komma und deshalb rechnet: $\tilde{r}$ = 1,3717420 !

Definition 3:
Die Zahl $f_a = |\tilde{r} - r|$ heisst **absoluter Rundungsfehler.**
In unserem Beispiel $f_a = 1{,}371742.10^8$

Die Zahl $f = \frac{f_a}{r}$ heisst **relativer Rundungsfehler.**

Im Beispiel $f = 10^8$

Werkstatt Multiplikation Theorie
Posten:**Rundungsfehler**

Es gibt eine sehr wichtige Masseinheit um die Präzision eines Computers auszudrücken.
Wir können sagen, dass ein Computer desto präziser ist , desto kleiner die Zahl ist, die zu 1 summiert ein anderes Ergebnis als 1 ergibt.

Definition 4:
Die kleinste positive Zahl ($\in$ *IM* !), die zu 1 summiert ein Ergebnis $\neq$ 1 gibt, heisst **Maschinengenauigkeit** und wird $\square$ (*epsilon*) geschrieben.

$$\square = \min \{z \in IM \mid z > 0 \; and \; 1 + z \neq 1\}$$

Als Beispiel $\square$ von Matlab auf Sun ist $2,22 \cdot 10^{-16}$.

Ein spezieller Rundungsfehler tritt auf, wenn zwei nahezu gleichgrosse reelle Zahlen substrahiert werden:
Betrachten wir folgende Subtraktion :

$$\begin{array}{l} 1,2345 - \\ 1,2344 \\ \hline 0,0001 = 1,0000 \cdot 10^{-4} \end{array}$$

Diese Nullen sind nur richtig, wenn die Operanden exakt waren!
Wenn diese Operanden hingegen schon gerundet wurden, dann ist das Ergebnis nicht mehr richtig! (Das exakte Ergebnis wäre 1, D ... D . 10^{-4}, D = eine Ziffer zwischen 0 ... 9). Der relative Rundungsfehler wird in einem solchen Fall sehr gross (weil das Ergebnis klein ist).
Diese Art Rundungsfehler heisst **numerische Auslöschung**.
Manchmal kann man die numerische Auslöschung dank Umformungen in den Rechnungen vermeiden.
Zum Beispiel sollen wir $\frac{1}{1-\sqrt{1-x^2}}$ für kleine x berechnen.

Problem: falls $\sqrt{1-x^2} \simeq 1 \quad \Rightarrow \quad \frac{1}{1 \simeq 1} \quad \Rightarrow$ Gefahr von numerische Auslöschung im Nenner!

D.h. falls $\sqrt{\varepsilon} > x$ ist, wird der Nenner gleich Null!
Hier kann folgende Umformung helfen:

$$\frac{1}{1-\sqrt{1-x^2}} \cdot \frac{1+\sqrt{1-x^2}}{1+\sqrt{1-x^2}} = \frac{1+\sqrt{1-x^2}}{x^2}$$

So kann auch der Fall $\sqrt{\varepsilon} > x$ ausgewertet werden.

Werkstatt Multiplikation Theorie
Posten:**Rundungsfehler**

Zusammenfassung:
Wir haben die Zahlen, mit denen der Computer arbeitet, kennengelernt und gesehen, wie sie die reellen Zahlen darstellen. Diese werden durch die Maschinenzahlen gerundet.
Wir haben gelernt, dass die Computer Grenzen haben. Wenn diese überschritten werden, kommt es zu Fehlern wie der Overflow- oder der Underflowerror.
Nicht immer aber behindern uns diese Fehler in der Ausführung unserer Rechnungen: einige Situationen von Overflow und von Underflow können gelöst werden.
Wir haben ausserdem gesehen, wie sich die impliziten Rundungen mit den Maschinenzahlen auf die mathematischen Operationen und ihre Lösungen auswirken. An dieser Stelle wurden die Rundungsfehler und die Maschinengenauigkeit definiert.
Der numerische Informatiker programmiert den Computer so, dass er die richtige Nährungslösung ausrechnet. Weiter ist es seine Aufgabe, zu verstehen weshalb ein Algorithmus schlecht funktioniert und ihn dann zu korrigieren.
Abschliessend noch ein Ratschlag: wenn du deine Programme wirklich testen willst, musst du sie mit Grenzfällen (das sind Werte, die an der Grenze der Computermöglichkeit liegen) prüfen!