Einführung in die Programmiersprache „BASIC" (Teil 2)

Maximilian Herberger

2. Probleme der Rechengenauigkeit

2.1. Begründung der Fragestellung

Im ersten Teil dieser Serie war abschließend vom Problem der Rechengenauigkeit die Rede. Auch Friederici hat auf die damit zusammenhängenden Probleme hingewiesen (s. IuR 1/1986, S. 39). Da es sich um ein für die Beurteilung des Rechnereinsatzes und für die eigene Programmierpraxis fundamentales Problem handelt, werden in den weiteren Folgen im Anschluß an die Erläuterung einiger Grundbegriffe zunächst drei kleine Beispielsprogramme vorgestellt, die diese Problematik veranschaulichen und gleichzeitig eine erste Vorstellung davon vermitteln, worum es sich bei einem „Programm" handelt. Das erlaubt es dann auch, das Verhältnis zwischen Eigenarten von Programmiersprachen und bestimmten juristischen Bewertungen (etwa im Urheberrecht) oder Anforderungen (z.B. im Vertragsrecht) in einer ersten Annäherung zu diskutieren. Außerdem läßt sich an Hand dieser Beispiele einer der ersten Fälle von Computerkriminalität besser verstehen. Das gewählte Thema ist zusätzlich deswegen als Einstieg besonders geeignet, weil bestimmt auch unter vielen Juristen die Meinung vorherrscht, ein „Rechner" sei dadurch gekennzeichnet, daß er „rechne" und dies mit einer ungeheuren Exaktheit tue. Wie sich zeigen wird, bedarf diese Einschätzung, soweit die Exaktheit betroffen ist, einer Einschränkung.

2.2. Grundbegriffe

2.2.1. Das Konzept der Variablen und die Zuweisungsoperation

Eine Variable kann man sich am besten als Namen vorstellen, unter dem ein bestimmter Speicherplatz im Rechner für Werte reserviert ist. Oder eine Nuance anschaulicher ausgedrückt: Ein „Etikett" auf einer „Schublade", in der bestimmte „Gegenstände" abgelegt werden. Wie sieht dieser Vorgang nun im einzelnen aus? Angenommen, man habe einer Variablen den Namen „zahl" gegeben und ihr den Wert 1 zugewiesen. Dann geschieht folgendes: Der Computer reserviert unter der Bezeichnung „zahl" einen Speicherbereich und legt dort den Wert 1 ab. Leider hat man in BASIC (wie auch in anderen Programmiersprachen) für die Darstellung dieses Vorgangs eine außerordentlich mißverständliche Schreibweise gewählt, die folgendermaßen aussieht:

```
10 zahl = 1
```

(10 ist eine Zeilennummer. BASIC verlangt diese Zeilennummern. Dazu sogleich Näheres bei dem ersten Beispielsprogramm).

Das Gleichheitszeichen steht hier für die Zuweisung des Wertes 1 an die Variable „zahl". Warum das zu Mißverständnissen führen kann, sieht man hier noch nicht, da ja „zahl" in einem gewissen Sinne „gleich" 1 ist. Eine weitere Überlegung macht aber sofort klar, warum es besser wäre, nicht das Gleichheitszeichen zu verwenden. Angenommen, man habe die Absicht, in einem zweiten Schritt den eben der Variablen „zahl" zugewiesenen Inhalt um 2 zu erhöhen. Dann muß man dem durch „zahl" bezeichneten Speicherplatz den vorherigen Inhalt der Variablen „zahl" um 2 erhöht zuweisen. In BASIC sieht das so aus:

```
20 zahl = zahl + 2
```

Zu lesen wäre das Ganze (von rechts nach links) so:

Nimm 2, erhöhe den bisherigen Inhalt der Variablen „zahl" (nämlich 1) um diesen Betrag von 2 und weise das Ergebnis (nämlich 3) der Variablen „zahl" zu, die damit den Wert 3 erhält.

Es leuchtet ein, daß etwa eine Schreibweise wie die folgende (mit einem nach links gerichteten Pfeil) das besser veranschaulichen würde:

```
zahl ← zahl + 2
```

Denn mathematisch gelesen wäre „zahl = zahl + 2" wegen fehlender Identität zwischen linker und rechter Seite der Gleichung ja sogar falsch. Gemeint ist aber nicht diese mathematische Identität, sondern die beschriebene Zuweisungsoperation, so daß das Gleichheitszeichen stets in diesem Sinne „von rechts nach links" zu lesen ist.

2.2.2 Variablenarten: Numerische Variable und Stringvariable

In dem eben behandelten Beispiel war „zahl" eine numerische Variable. Das bedeutet, daß man ihr Zahlenwerte zuweisen kann. Daneben gibt es noch sogenannte „Stringvariable". Diesen Variablen werden Zeichenketten zugewiesen. Gekennzeichnet werden diese Stringvariablen dadurch, daß man hinter den gewählten Namen ein „$" setzt und die ihnen zugewiesene Zeichenkette durch Anführungszeichen kennzeichnet, also z. B. so:

```
30 bezeichnung$ = "computer"
```

Als Ergebnis dieser Zuweisungsoperation steht in dem durch den Variablennamen „bezeichnung$" gekennzeichneten Speicherbereich die Zeichenkette „computer". Übrigens können Zahlen Bestandteil einer solchen Zeichenkette sein. Ein Beispiel wäre das folgende:

40 bezeichnung$ = "computer1"

In diesem Fall wird die „1" nicht als Zahl behandelt, sondern einfach „wie ein Buchstabe" als Bestandteil der Zeichenkette „computer1".

2.3. Beispielsprogramme zum Thema „Rechengenauigkeit"

2.3.1 Das Programm PROG0.BAS

2.3.1.1 Mathematische Problemstellung

Das Programm PROG0.BAS demonstriert den Unterschied der Ergebnisse bei zwei verschiedenen Berechnungsarten für dasselbe numerische Problem und veranschaulicht auf diese Weise, welche Rechenungenauigkeit in bestimmten Situationen auftreten kann. Das elementare Problem ist kurz gesagt folgendes: Jemand möchte bestimmen, zu welchem Ergebnis es führt, wenn er von 0 ausgehend zehnmal 0,1 zu dem durch die jeweilige Addition entstehenden Wert hinzuaddiert. Mathematisch ist der Fall klar:

Übersicht 1

Schritt	Ausgangswert	Rechnung	Zwischenergebnis
1	0,0	0,0+0,1	0,1
2	0,1	0,1+0,1	0,2
3	0,2	0,2+0,1	0,3
4	0,3	0,3+0,1	0,4
5	0,4	0,5+0,1	0,5
6	0,5	0,5+0,1	0,6
7	0,6	0,6+0,1	0,7
8	0,7	0,7+0,1	0,8
9	0,8	0,8+0,1	0,9
10	0,9	0,9+0,1	1,0

Die jeweiligen Zwischenergebnisse müssen genau so aussehen, wie sie in der Spalte „Zwischenergebnisse" erscheinen: 0,6 + 0,1 ist exakt 0,7. Wer etwa 0,7000001 als Ergebnis behaupten würde, würde einen Fehler machen.

Neben der eben dargestellten Methode ist noch eine zweite zur Berechnung der einzelnen Zwischenergebnisses denkbar. Statt schrittweise zu addieren könnte man in jedem Schritt 0,1 mit der für den jeweiligen Schritt stehenden Zahl multiplizieren und dieses Produkt zu dem Ausgangswert addieren. Das sähe etwa für Schritt 7 so aus: 0,1 mal 7 = 0,7. Dieses zu dem Ausgangswert 0 addiert ergibt 0,7. Diese zweite Methode muß zu genau denselben Ergebnissen führen wie die erste Methode.

Das folgende Programm rechnet nach den beiden eben beschriebenen Methoden und vergleicht die Ergebnisse miteinander. Bevor diese Ergebnisse diskutiert werden, wird jedoch zuerst das Programm im Quellcode (das sogenannte „Listing") abgedruckt und dann zeilenweise erläutert. Bei der Erläuterung wird ergänzend auf das Programm von Friederici hingewiesen, wenn einzelne der kommentierten Befehle dort vorkommen. Das erlaubt es dem Leser, schrittweise dieses Programm nachzuvollziehen.

2.3.1.2 Listing des Beispielsprogramms 1: PROG0.BAS

```
10 REM Programmkopf
20 REM Programmname: PROG0.BAS
30 REM Datum: 21.3.86
40 REM Autor: N.N.
50 REM Beginn Hauptprogramm
60 PRINT "Methode 1", "Methode 2",TAB(39)"Differenz" 'Drucken der Ueberschrift
70 PRINT                'Drucken einer Leerzeile
80 FOR I=1 TO 10        'Beginn einer Schleife mit 10 Durchgaengen
90     J=J+.1           'Erhoehen des Wertes der Variablen J um 0.1
100    PRINT J,         'Ausdrucken von J
110    S=I*.1           'Berechnen von S
120    PRINT S,         'Ausdrucken von S
130    D=S-J            'Berechnen von D
140    PRINT D;         'Ausdrucken von D (Exponentialschreibweise)
150    PRINT TAB(50) USING "#.###############";D 'Ausdrucken von D (formatiert)
160 NEXT I              'Ende der Schleife
170 STOP                'Programmende
```

2.3.1.3 Kommentar zum Listing von PROG0.BAS

Ein BASIC-Programm ist im Quell-(Source-)Code eine Folge von Textzeilen, die mit fortlaufenden Zeilennummern versehen sind. Der BASIC-Interpreter interpretiert diese Zeilen (mit Ausnahme der REM-Zeilen, dazu sogleich) als Befehle und führt diese Befehle aus.

Übersicht 2: Erläuterte Elemente der Programmiersprache BASIC

FOR ... TO und NEXT
PRINT
PRINT TAB
PRINT USING
REM
STOP

Zeilen 10–40:

10 REM Programmkopf
20 REM Programmname: PROG0.BAS
30 REM Datum: 21.3.86
40 REM Autor: N.N.

Diese mit REM (= Remark) eingeleiteten Zeilen werden vom Programm nicht als ausführbare Befehlszeilen interpretiert. Man benutzt diese Möglichkeit, um das Programm zu kommentieren und so lesbarer zu machen. Außerdem erleichtert eine geschickte Handhabung dieser Technik das Wiedereinarbeiten in Programme, mit denen man lange nicht mehr gearbeitet hat. Hier wurde, was üblich ist, ein Programmkopf mit wichtigen Angaben zum Programm vorangestellt. Häu-

fig wird diese Stelle auch benützt, um einen Copyright-Vermerk anzubringen. Für die juristische Beurteilung des Schutzes von Programmen ist von Bedeutung, daß solche Teile des Quellprogrammes normaler Text sind, wie er beispielsweise auch in einer Programmdokumentation erscheinen könnte. Es ist also möglich, einen regelrechten Programmkommentar als Teil des Quellprogramms zu schreiben.
(Vgl. bei Friederici z. B. Zeile 4000 u.ö.; IuR 2/1986, S. 90).

Zeile 50:

50 REM Beginn Hauptprogramm

Es ist günstig, einzelne Teile eines Programms durch eine REM-Zeile zu kennzeichnen, um so die Struktur des Programms hervortreten zu lassen.
(Vgl. bei Friederici z. B. Zeile 230, 260, 280 u.ö.; IuR 2/86, S. 91).

Verschiedentlich arbeiten Programmautoren zusätzlich noch mit graphischen Techniken (Leerzeilen, Zeilen mit „*" etc.), um diese Markierungen zu unterstreichen. Eine Lösung dieser Art wäre etwa die folgende:

```
45 REM  ******************
50 REM  * Hauptprogramm *
55 REM  ******************
```

Da diese Gestaltungsmöglichkeiten bestehen, vertreten manche Autoren die Ansicht, daß Programme im Quellcode ähnlich wie etwa Tabellen oder Fahrpläne das Ergebnis von Überlegungen zur Anordnung sind, die mit dem Ablauf des Programms, seinem technischen Funktionieren nichts zu tun haben, und demnach eine eigene gestalterische Leistung des Programmautors sind. Das gilt in verstärktem Maße für die Kommentare zum Programm. Die entsprechende Qualität könnte man die „Lesbarkeit von Programmen" nennen. Der BGH hat diesen Aspekt in seinem Urteil zum Urheberrechtsschutz für Computerprogramme nicht berücksichtigt. Dort heißt es nur: Die „Kodierung wird in der Regel zunächst unabhängig von der Maschinensprache des zur Verfügung stehenden Computers in einer Programmiersprache vorgenommen; das Ergebnis ist das für den Fachmann lesbare sogenannte Primär- oder Quellenprogramm. Durch maschinelle Übersetzung des Quellenprogramms entsteht das sogenannte Objektprogramm, das der Maschinensprache direkt entspricht" (IuR 1/1986, S. 20). Daß diese Überlegung unvollständig ist, erkennt man an folgendem: Bei der Übersetzung des Quellprogramms werden die kommentierenden Anmerkungen nicht in Befehle umgewandelt (was ja auch gar nicht möglich wäre). Sie bilden also einen deutlich eigenständigen Teil, der deswegen auch eine selbständige Beurteilung verdient.

Zeile 60:

60 PRINT "Methode 1","Methode 2",TAB(39)"Differenz" 'Drucken der Ueberschrift

Der Befehl PRINT gibt die in Anführungszeichen gesetzten Texte (= Stringvariablen) auf dem Bildschirm aus (vgl. z. B. bei Friederici Zeilen 40, 50, 60 u.ö., IuR 2/1986, S. 91). Dabei kann man verschiedene Ausgabeformate erzielen. So bewirkt beispielsweise das Komma zwischen „Methode 1" und „Methode 2", daß der Text „Methode 2" an der Tabulatorstelle 15 beginnt. (Anders ausgedrückt: Ein Komma verschiebt den Ausgabebeginn jeweils um 14 Stellen.) Eine andere Methode wird durch das TAB(39) demonstriert. Dieser Zusatz bewirkt, daß das Wort „Differenz" beginnend mit Tabulatorstelle 39 gedruckt wird. Der Text nach dem Apostroph (hier: „Drucken der Überschrift") ist wieder ein Kommentar. Man kann damit in einer Zeile kommentieren.

Zeile 70:

70 PRINT 'Drucken einer Leerzeile

PRINT ohne Argument, d. h. ohne Angabe einer zu druckenden Variablen, druckt eine Leerzeile auf dem Bildschirm.
(Vgl. bei Friederici z. B. Zeile 60, IuR 2/1986, S. 91. Dort ist zu beachten, daß man in BASIC nach einem Doppelpunkt in derselben Zeile weiterschreiben und so einen neuen Befehl geben kann. Das dient hauptsächlich der Platzersparnis.)

Mit dem PRINT-Befehl und anderen Befehlen zur Ausgabe auf den Bildschirm, laesst sich ein visueller Eindruck gestalten, der den Programmablauf auf dem Bildschirm begleitet. Mit Graphik- und Farbbefehlen kann das bis zur Darstellung regelrechter Bilder führen. Hier liegt ein weiteres Diskussionsthema für das Immaterialgüter- und das Wettbewerbsrecht: Die optische Bildschirmgestaltung und Benutzerführung auf dem Bildschirm könnte ab einem gewissen Grad Schutz verdienen. In den USA gibt es bereits einen Beispielsfall dieser Art. „Apple" und „Digital Research" einigten sich im Vorfeld eines Prozesses darauf, daß „Digital Research" eines seiner Programme optisch, der Darstellung auf dem Bildschirm nach deutlicher als bisher von einem vergleichbaren Apple-Produkt absetzt.

Zeile 80:

80 FOR I=1 TO 10 'Beginn einer Schleife mit 10 Durchgaengen

Zeile 80 bildet mit Zeile 170 eine sogenannte „Schleife". In einer Schleife wird eine bestimmte Operation mehrfach wiederholt. Hier geschieht dies zehnmal, da die Zählvariable I von 1 bis 10 läuft. Anders ausgedrückt: Das Programm zählt intern von 1 bis 10 und führt solange den Schleifeninhalt aus. Es kehrt gewissermaßen während der so definierten Wiederholungsdauer immer wieder von Zeile 170 zu Zeile 80 zurück.
(Vgl. bei Friederici z.B. Zeilen 490 und 510, IuR 2/1986, S. 91).

Schleifen sind als Wiederholoperationen eines der wichtigsten Programmierelemente. Der BGH hat dies in dem bereits zitierten Urteil auch dementsprechend als ein Charakteristikum berücksichtigt (vgl. IuR 1/1986, S. 20).

Zeile 90:

```
90 J=J+.1 'Erhoehen des Wertes der Variablen J um 0.1
```

Hier liegt eine Zuweisungsoperation der oben (vgl. 2.1.) beschriebenen Art Zu der bei Programmanfang auf 0 stehenden Variablen J wird 0,1 addiert und dieser Wert (nach dem ersten Schritt 0,1) wird der Variablen J zugewiesen. (In BASIC schreibt man 0.1 für 0,1. Der Punkt steht für das Komma.) Was das Programm in den einzelnen Wiederholschritten tut, deckt sich dem Ablauf nach mit der oben gegebenen Übersicht 1. (Vgl. bei Friederici z. B. die Zeilen 580 und 590, IuR 2/1986, S. 91. Die dort verwendete besondere Variablenart wird später erläutert.)

In dem Programm sind die in der Schleife vorkommenden Befehle eingerückt geschrieben. Wenn man das systematisch tut, wird das Programm leichter lesbar, weil seine Struktur deutlicher hervortritt.

Zeile 100:

```
100 PRINT J,  'Ausdrucken von J
```

Ausdruck des jeweiligen Wertes von J. Das Komma nach J bewirkt, daß der nächste Bildschirmausdruck nicht in einer neuen Zeile beginnt, sondern an Tabulatorposition 15 in derselben Zeile. Allgemein läßt sich die Wirkungsweise des Kommas folgendermaßen beschreiben: Das Komma rückt immer auf eine Schreibstelle vor, die dem nächsten Vielfachen von 15 entspricht. Wird dabei die Position 80 überschritten, wird der Ausdruck in Position 1 der folgenden Zeile fortgesetzt.

Zeile 110:

```
110 S=I*.1 'Berechnen von S
```

In dieser Zeile wird nach der oben beschriebenen zweiten Methode gerechnet: In jedem Schritt wird 0,1 (der BASIC-Interpreter wandelt die Eingabe 0.1 in .1 um) mit der für den jeweiligen Schritt stehenden Zahl (= der jeweilige Wert der Zählvariablen I) multipliziert. (Der Stern ist in BASIC das Multiplikationszeichen.) Das Ergebnis dieser Multiplikation wird der Variablen S (für Summe) zugewiesen. Es ist übrigens nicht notwendig, die Variablenbezeichnungen bis auf einen Buchstaben abzukürzen. Wenn man längere Namen wählt, kann man auch auf diese Weise das Programm in einer „sprechenden" Weise gestalten und lesbar machen.

Zeile 120:

```
120 PRINT S,  'Ausdrucken von S
```

Ausdruck des jeweiligen Wertes von S. Das Komma nach S hat zur Folge, daß der nächste Wert an Tabulatorposition 29 in derselben Zeile gedruckt wird. (Vgl. die Erläuterung zu Zeilen 60, 100).

Zeile 130:

```
130 D=S-J  'Berechnen von D
```

Zwischen S und J dürfte aus mathematischen Gründen kein Unterschied bestehen. Um zu testen, ob das so ist, wird die Differenz D aus beiden Werten gebildet. D müßte nach den Eingangsüberlegungen stets 0 sein.

Zeile 140:

```
140 PRINT D; 'Ausdrucken von D (Exponentialschreibweise)
```

D wird am Bildschirm ausgedruckt. Das Semikolon nach D bewirkt, daß ein weiterer Ausdruck nicht in einer neuen Zeile beginnt, sondern in der laufenden Zeile fortgesetzt wird.

Zeile 150:

```
150 PRINT TAB(50) USING
"###.#######",D  'Ausdrucken von D (formatiert)
```

D wird am Bildschirm beginnend an der Tabulatorposition 50 ausgedruckt. Dabei wird ein besonderes Druckformat verwendet, das aus bis zu drei Vorkommastellen und sieben Nachkommastellen besteht. Der Zusatz USING zu PRINT bringt dieses Druckformat zum Ausdruck. Verwendet man den PRINT USING-Befehl, so ist die letzte Stelle innerhalb des Formats eine auf Grund der in der zu druckenden Zahl nachfolgenden Ziffern gerundete Zahl.

Zeile 160:

```
160 NEXT I  'Ende der Schleife
```

Hier ist das Ende der in Zeile 80 beginnenden Schleife.

Zeile 170:

```
170 STOP 'Programmende
```

STOP markiert das Programmende.

2.3.1.4 Diskussion der Ergebnisse von PROG0.BAS

Was erscheint nun am Bildschirm, wenn man das Programm ablaufen läßt? Das Ergebnis eines Programmlaufs ist in Übersicht 3 zu sehen.

Bevor dieser Ausdruck interpretiert wird, bedürfen die auf den ersten Blick merkwürdigen Zahlen einer Erläuterung, die mit „E-08" bzw. „E-07" enden. Sie sind folgendermaßen zu lesen:

Übersicht 3: Vom Programm PROG0.BAS erzeugter Ausdruck

Methode 1	Methode 2	Differenz	
.1	.1	0	0.0000000
.2	.2	0	0.0000000
.3	.3	0	0.0000000
.4	.4	0	0.0000000
.5	.5	0	0.0000000
.6	.6	0	0.0000000
.7000001	.7	-5.960465E-08	-0.0000001
.8000001	.8	-5.960465E-08	-0.0000001
.9000001	.9000001	-5.960465E-08	-0.0000001
1	1	-1.192093E-07	-0.0000001

-5.960465E-08 entspricht
-5,960465 mal (das E steht für „Exponent") 10 hoch minus acht (was wiederum -5,960465 durch 10 hoch acht und damit 0,00000005960465 entspricht. Das ist bis zur siebten Nachkommastelle aufgerundet -0,0000001, was in der rechten Spalte erscheint.)

Warum diese Darstellungsweise? Der Computer läßt uns hier einen Blick in sein Inneres tun, auf die Art und Weise nämlich, wie er Zahlen darstellt. Alle Zahlen sind im binären Zahlensystem ausgedrückt und zwar so, wie man es in der obigen Tabelle für das Dezimalsystems sieht: Eine sogenannte Mantisse (hier: -5,960465) wird mit einer Potenz (hier 10 hoch minus 8) multipliziert. Diese Darstellungsform ist auch dafür verantwortlich, daß es eine Genauigkeitsgrenze für die im Rechner darstellbaren Zahlen gibt. Denn alle Zahlen werden znächst in das Dualsystem tranformiert. Dort werden sie in der ebenen beschriebenen Form mit Mantisse und Exponent gespeichert. Nun ergeben aber einige Zahlen im Dualsystem nicht abbrechende Dualbrüche. Diese können dann mit einer der Länge nach begrenzten Mantisse nicht vollständig exakt dargestellt werden. Die Länge der Mantisse bildet damit die Obergrenze für die Genauigkeit: Je länger die Mantisse, desto größer die Rechengenauigkeit. Es existiert dafür allerdings eine Obergrenze und damit eine prinzipielle Schranke der Rechengenauigkeit. Diese kann man in größeren Rechnern so weit hinausschieben, daß sie praktisch keine Rolle mehr spielt. Das ändert aber nichts an der prinzipiellen Begrenzung, die auch dort vorhanden ist. Für die Programmiersprache BASIC begegnet man der mit dieser Darstellungsweise zusammenhängenden Ungenauigkeit in dem vorliegenden Beispiel. (Dabei ist allerdings der Vollständigkeit halber hinzuzufügen, daß nicht alle BASIC-Versionen in gleicher Weise reagieren. Hier wurde das mit dem IBM Personal Computer XT ausgelieferte BASIC 3.0 verwandt. Das BASICA von Microsoft zeigt die gleiche Ungenauigkeit. Bessere Ergebnisse lieferte beispielsweise die neue Basic-Version BETTER BASIC. Beim Ablaufen von PROG0.BAS unter BETTER BASIC sind die ersten beiden Spalten vollständig korrekt. In der vierten Spalte ist die Differenz in den ersten neuen Schritten auf sieben Stellen genau. Bereits die Auswahl der BASIC-Version kann also einen grossen Unterschied machen, und die hier vorgestellten kleinen Testprogramme sind geeignet, erste Hinweise zu geben).

Bis zum sechsten Schritt entspricht alles den Erwartungen: Methode 1 und Methode 2 führen zu denselben Ergebnissen, die Differenz ist Null. Ab dem siebten Schritt ändert sich aber das Bild und zeigt einige erstaunliche Ergebnisse. Bei Methode 1 taucht im siebten, achten und neunten Schritt in der siebten Nachkommastelle eine Ungenauigkeit auf, bei Methode 2 ist das im neunten Schritt der Fall. Die Differenzen schließlich zeigen, daß man es in den Spalten „Methode 1" und „Methode 2" mit dem Ergebnis von Rundungsvorgängen zu tun hat. Denn diese Differenzen würden sich so nicht ergeben, wenn man mit den in den beiden ersten Spalten angegebenen Zahlen rechnen würde. Der Rechner tut also etwas, was nicht vollständig den Regeln mathematischer Präzision entspricht.

2.3.1.5 Einige Anmerkungen aus der juristischen Perspektive

Wenn es eine prinzipielle Grenze der Rechengenauigkeit gibt, wird man von einem Programmierer im Rahmen der ihm obliegenden Sorgfaltsanforderungen nicht mehr fordern können, als der Rechner (mit der Programmiersprache) zuläßt. Es handelt sich schlicht und einfach um einen Anwendungsfall der Maxime „impossibilium nulla obligatio est".

Allerdings zeigt schon das kleine, anspruchslose Programm PROG0.BAS, daß bei Anwendung verschiedener Methoden (im Rahmen der durch den Rechner und die Programmiersprache gezogenen Grenzen) unterschiedlich gute Ergebnisse zu erzielen sind: Wenn es auf die siebte Nachkommastelle ankommt, verdient Methode 2 in dem Beispiel den Vorzug vor Methode 1. Es entspräche dann nicht den an einen Programmierer zu stellenden Sorgfaltsanforderungen, wenn dieser Methode 1 verwenden würde. Auch ist zu berücksichtigen, daß es sehr einfallsreiche Hilfsprogramme gibt, die es erlauben, die Rechengenauigkeit über die standardmäßig gegebenen Grenzen einer Programmiersprache hinauszuschieben. Meistens kann man damit in Bereiche vorstossen, die der Problematik ihre praktische Bedeutung nehmen.

Unter Umständen können die Sorgfaltspflichten aber sogar schon vor dem Programmieren beginnen, bei der Auswahl der Programmiersprache nämlich. Denn die Rechengenauigkeit ist in den einzelnen Programmiersprachen unterschiedlich. Je nach Problemstellung wird dann u. U. die größere Genauigkeit erlaubende Programmiersprache zu verwenden sein.

Ein besonderes Problem entsteht, wenn der Gesetzgeber Rechengrößen vorgibt, die in den kritischen Bereich der normalen Rechengenauigkeit hineinreichen. Bei den Faktoren aus der Rechengrößenverordnung, die im Rahmen des Versorgungsausgleichs anzuwenden ist, verhält es sich so (vgl. Friederici, IuR 1/1986, S. 39). Es gilt dann, wenn der Einsatz der EDV aus guten Gründen zu befürworten ist, Programmiertechniken zu wählen, die für die gesetzgeberische Anforderung einen Genauigkeitsbereich schaffen.

(wird fortgesetzt)