Auch die Wege zu einem zentralen Drucker kosten viel Personalkosten, so daß sich insbesondere bei weitläufigen Anwaltskanzleien unbedingt der Einsatz lokaler Arbeitsplatzdrucker empfiehlt. Ein optimales Computersystem sollte die Möglichkeit enthalten, zusätzlich zu zwei zentralen Druckern, die von allen Arbeitsplätzen aus angesteuert werden (z.B. einen Schönschreib-Typenraddrucker mit Zweikammern-Einzelblatteinzug für die normale Korrespondenz und einen Matrixdrucker mit Endlospapier für Buchhaltung und Zwangsvollstreckung), noch lokale Drucker an jedem Arbeitsplatz anzuschließen, vorzugsweise eine elektronische Typenradschreibmaschine mit einer Schnittstelle. Hier gibt es schon sehr preiswerte Geräte ab 1300,00 DM. Der Sachbearbeiter könnte diese dann sowohl als Schreibmaschine für das eine oder andere verwenden, das nicht über den Computer laufen soll, oder kann z.B. das Ausfüllen mancher Formulare und auch den Ausdruck von Mahnbescheiden erledigen; denn bei der Entscheidung für einen oder mehrere Drucker muß eben auch die Möglichkeit des Ausdruckes derartiger Formularsätze bedacht werden. Aus praktischen Erwägungen ist jede Lösung abzulehnen, die etwa vorsieht, daß derartige Aufträge gespeichert werden und dann z.B. täglich ab 16.00 Uhr in einen gemeinsamen Drucker die Mahnbescheidsformulare eingespannt und dann ausgedruckt werden. Dies funktioniert in der Regel nur theoretisch. Ein gemeinsamer Drucker in einer Mehrplatzanlage muß, um Störungen im Betriebsablauf zu vermeiden, stets in gleicher Weise für alle zur Verfügung stehen — es geht nicht, daß ein Einzelner beispielsweise den Einzelblatteinzug gelegentlich außer Betrieb setzt und einen Mahnbescheid ausdruckt, während der andere gerade den noch vor 18.00 Uhr zur Post müssenden Schriftsatz ausdrucken will.

# Einführung in die Programmiersprache „BASIC" (Teil 4)

**Maximilian Herberger**

Nachdem in den vorigen Folgen am Beispiel von Programmen zur Rechengenauigkeit verschiedene elementare Operationen und Grundkonzeptionen vorgestellt wurden, beginnt heute die systematische Erörterung einzelner notwendiger Programmteile. Ein Programm gliedert sich in der Regel in

— Eingabeteil,
— Operationsteil,
— Ausgabeteil.

Im Eingabeteil erhält das Programm die für die Arbeit notwendigen Daten. Eingaben sind möglich

— über die Tastatur,
— aus Dateien („Files"),
— über einen Kommunikationskanal (von anderen Rechnern oder Peripheriegeräten).

Im Operationsteil verarbeitet das Programm die Eingaben. Die Ergebnisse werden dann im Ausgabeteil ausgegeben. Dabei sind Ausgaben denkbar

— über den Bildschirm,
— in Dateien („Files"),
— auf einen Kommunikationskanal (zu anderen Rechnern oder Peripheriegeräten).

Die sich aus dieser Übersicht ergebenden Einzelkomplexe werden im weiteren Verlauf dieser Serie nacheinander behandelt. Die heutige Folge beschäftigt sich mit der Dateneingabe über die Tastatur.

(Die Auflösungen zu den Übungen aus der vorherigen Folge befinden sich am Ende dieses Teils der Serie.)

Übersicht 12: Erläuterte Elemente der Programmiersprache BASIC
ASC
CHR$
VAL

## 3. Formen der Dateneingabe

### *3.1 Die Dateneingabe über die Tastatur*

#### *3.1.1 Vorüberlegung zur juristischen Relevanz von Eingabeprüfungen*

Der Grundbefehl für die Dateneingabe über die Tastatur wurde bereits in der vorigen Folge behandelt. Zeile 30 in PROG1.BAS lautete:

30 INPUT „Geben Sie bitte eine Ziffer ein!", ZAHL

Dieser Befehl läßt den in Anführungszeichen stehenden Text auf dem Bildschirm erscheinen und wartet eine Eingabe ab. Eine eingegebene Ziffer wird dann der Variablen ZAHL als Wert zugewiesen.

Will man ein gutes Programm schreiben, ist es aber mit dieser Zeile nicht getan. Vielmehr muß man bei Eingaben über die Tastatur verschiedene Eingabekontrollen vorsehen. Diese Kontrollen sollen für den Fall Vorsorge treffen, daß der Benutzer bei der Eingabe Fehler macht. Dabei ist es das Ziel, diese Fehler möglichst benutzerfreundlich abzufangen. Die Art und Weise, wie Fehler abgefangen werden, ist ein wichtiges Softwarequalitätskriterium, auf das deshalb bei Software-Tests auch besonders geachtet wird.

Juristisch betrachtet handelt es sich um einen Software-Mangel, wenn ein voraussehbarer Eingabefehler nicht zurückgewiesen wird. Das klingt in dieser Form vielleicht noch sehr theoretisch. Es soll aber bereits hier darauf hingewiesen werden, daß nicht alle Kanzleiverwaltungspakete diese Anforderung vollständig erfüllen. Die folgenden Überlegungen sind deshalb auch eine Anleitung dazu, wie man angebotene Programme unter dem hier behandelten Gesichtspunkt „destruktiv" testen kann.

*3.1.2 Denkbare Eingabefehler*

*3.1.2.1 Eingabe einer anderen Datenart als erwartet*

*3.1.2.1.1 Eingabe eines Buchstabens statt einer Ziffer*

Ein besonders häufiger Eingabefehler ist der, daß nicht die im Programm erwartete Datenart eingegeben wird. Bleiben wir bei dem Beispiel der Eingabezeile aus PROG1.BAS:

```
30 INPUT „Geben Sie bitte eine Ziffer ein ! “,ZAHL
```

Die Variable ZAHL ist numerisch, d.h. es wird die Eingabe einer Ziffer erwartet. Was geschieht, wenn der Benutzer stattdessen einen Buchstaben eingibt? Dieser Fehler kann leicht vorkommen, vor allen Dingen dann, wenn der Buchstabe O statt der Ziffer 0 eingegeben wird.

Was geschieht also nun, wenn der Benutzer in der obigen Beispielszeile einen Buchstaben statt einer Ziffer eingibt? Auf dem Bildschirm erscheint die Meldung

```
?Redo from start
```

und das Programm erwartet mit einem Fragezeichen erneut die Eingabe einer Ziffer.

Die Meldung „?Redo from start“ ist eine in BASIC fertig vorgesehene Fehlermeldung, die Eingaben von Buchstaben für numerische Variablen verhindern soll. Man könnte sich also damit beruhigen, daß BASIC sich um diesen Fehler kümmert. Das wäre aber keine gute Lösung, da der Benutzer — hat er nicht Programmierkenntnisse — über das „?Redo from start“ normalerweise sehr zu rätseln haben wird. Der Gesichtspunkt der Benutzerfreundlichkeit erfordert es also, eine bessere Lösung zu finden. Diese bessere Lösung beruht auf einem kleinen Kunstgriff, zu dessen Erläuterung das folgende kurze Programmstück dienen soll:

```
30 INPUT „Bitte geben Sie eine Ziffer ein ! “,ZAHL$
40 IF ASC(ZAHL$)⟨48 OR ASC(ZAHL$)⟩57 THEN GOSUB 100:GOTO 30
50 STOP
100 REM Text fuer den Eingabefehler: Anderes Zeichen statt Ziffer
110 PRINT „Es muss eine Ziffer eingegeben werden.“
120 PRINT „Sie haben “;ZAHL$;„ eingegeben.“
130 RETURN
```

Zeile 40 enthält eine Bedingung, die wahr wird, wenn die Eingabe keine Ziffer ist. (Die Arbeitsweise dieser Bedingung wird sogleich näher erläutert.) Ist die Bedingung wahr (wurde also keine Ziffer eingegeben), so „springt“ das Programm in das in Zeile 100 beginnende Unterprogramm. Auf dem Bildschirm erscheint der dort in Anführungszeichen vorgesehene Text. Dabei teilt der Befehl in Zeile 120 dem Benutzer noch mit, was er (fehlerhaft) eingegeben hat. Nach Abschluß der Unterroutine kehrt das Programm an den Ausgangspunkt in Zeile 40 hinter dem GOSUB zurück. Dort findet es das „GOTO 30“, führt es aus und fordert nochmals zur Eingabe einer Ziffer auf. Das wiederholt sich solange, bis eine Ziffer eingegeben wird. Um das Ganze zu veranschaulichen, ist im folgenden die Bildschirmausgabe für den Fall dargestellt, daß ein „U“ eingegeben wurde:

```
Bitte geben Sie eine Ziffer ein ! ⟨U⟩⟨RETURN⟩
Es muss eine Ziffer eingegeben werden.
Sie haben U eingegeben. Bitte geben Sie eine Ziffer ein !
```

(Anmerkung: Die Spitzklammern kennzeichnen eine Eingabe. Eine Eingabe, hier die des Buchstabens U, wird durch Drücken der RETURN-Taste abgeschlossen. Die RETURN-Taste wird auch als ENTER-Taste bezeichnet.)

Was hier auf dem Bildschirm erscheint, genügt den Anforderungen, die man an die Fehlerbehandlung stellt. Diese soll nämlich drei Schritte umfassen:

1. Die Aufmerksamkeit des Benutzers auf das Erwartete lenken. (Das tut Zeile 110.)
2. Den Fehler identifizieren. (Das tut Zeile 120.)
3. Zum erneuten Tätigwerden auffordern. (Das tut Zeile 30 im jeweils erneuten Durchgang.)

(Vgl. zu diesen Prinzipien der Fehlerbehandlung Henry Simpson, A Human-Factors Style Guide for Program Design, in: Decker F. Walker/Robert D. Hess (Hrsg.), Instructional Software Principles and Perspectives for Design and Use, Belmont 1984, S. 130–142, insb. 136.)

Die vorgestellte Prüfroutine verhindert keinesfalls alle möglichen Fehleingaben. Sie erfüllt nur die Aufgabe zu erkennen, wenn anstatt einer Ziffer ein Buchstabe eingegeben wird. Es ist jetzt noch zu erläutern, wie die Prüfbedingung in Zeile 40 funktioniert. Diese Prüfbedingung verwendet die Funktion ASC, die den ASCII-Wert des ersten Zeichens eines Strings bestimmt (vgl. zu den Variablenarten „numerisch“ und „String“ 2.2.2, Teil 2, IuR 1986, S. 183f.). Die ASCII-Werte (ASCII steht für „American Standard Code for Information Interchange“) spielen beim Programmieren eine wichtige Rolle. Es handelt sich dabei um eine Zuordnung von Zahlenwerten zu Funktionen (wie z.B. Zeilenvorschub = Line Feed oder Wagenrücklauf = Carriage Return usw.) und zu Zeichen. Für die folgenden Überlegungen sind zunächst nur die ASCII-Werte von Zeichen in einem bestimmten Bereich interessant, so daß sich die Darstellung einstweilen darauf beschränkt. Die Entsprechungen von ASCII-Wert und Zeichen sehen für die Werte von 33 bis 126 folgendermaßen aus:

Übersicht 13:

Die ASCII-Werte von 33 bis 126 und die zugehörigen Zeichen

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 033 | 034 | 035 | 036 | 037 | 038 | 039 | 040 | 041 | 042 |
| ! | " | # | $ | % | & | ' | ( | ) | * |
| 043 | 044 | 045 | 046 | 047 | 048 | 049 | 050 | 051 | 052 |
| + | , | — | . | / | 0 | 1 | 2 | 3 | 4 |
| 053 | 054 | 055 | 056 | 057 | 058 | 059 | 060 | 061 | 062 |
| 5 | 6 | 7 | 8 | 9 | : | ; | ⟨ | = | ⟩ |
| 063 | 064 | 065 | 066 | 067 | 068 | 069 | 070 | 071 | 072 |
| ? | @ | A | B | C | D | E | F | G | H |
| 073 | 074 | 075 | 076 | 077 | 078 | 079 | 080 | 081 | 082 |
| I | J | K | L | M | N | O | P | Q | R |

(weiter S. 274)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 083 | 084 | 085 | 086 | 087 | 088 | 089 | 090 | 091 | 092 |
| S | T | U | V | W | X | Y | Z | Ä | Ö |
| 093 | 094 | 095 | 096 | 097 | 098 | 099 | 100 | 101 | 102 |
| Ü | ^ | _ | ' | a | b | c | d | e | f |
| 103 | 104 | 105 | 106 | 107 | 108 | 109 | 110 | 111 | 112 |
| g | h | i | j | k | l | m | n | o | p |
| 113 | 114 | 115 | 116 | 117 | 118 | 119 | 120 | 121 | 122 |
| q | r | s | t | u | v | w | x | y | z |
| 123 | 124 | 125 | 126 | | | | | | |
| ä | ö | ü | ß | | | | | | |

Mit der Funktion ASC kann man den ASCII-Wert der in der Tabelle enthaltenen Zeichen ermitteln. Hat etwa die Stringvariable ZAHL$ den Wert „U", so ergibt die Funktion ASC(ZAHL$) gemäß der ASCII-Tabelle den Wert 85. Damit sind wir in der Lage, die Funktion von Zeile 40 aus der obigen Prüfroutine zu verstehen. Diese Zeile lautete:

```
40 IF ASC(ZAHL$)‹48 OR ASC(ZAHL$)›57 THEN GOSUB 100:GOTO 30
```

Der Inhalt dieser Zeile läßt sich folgendermaßen übersetzen:

Wenn der ASCII-Wert des Inhalts der Stringvariablen ZAHL$ kleiner als 48 ist oder wenn der ASCII-Wert des Inhalts der Stringvariablen ZAHL$ größer als 57 ist, dann „springe" in die in Zeile 100 beginnende Unterroutine und dann gehe (nach Rückkehr aus dieser Unterroutine) in Zeile 30.

Wenn man jetzt noch einmal die Tabelle der ASCII-Werte heranzieht, erkennt man, daß die Bedingung immer dann (und nur dann) wahr wird, wenn die Eingabe keine Ziffer ist. Denn die Ziffern haben die ASCII-Werte 48 bis 57. „Kleiner 48 oder größer 57" ist also gleichbedeutend mit „Keine Ziffer". Die Prüfroutine läßt deshalb nur Ziffern passieren.

An dieser Stelle sei des systematischen Zusammenhangs wegen noch die zur Funktion ASC „spiegelbildliche" Funktion CHR$ vorgestellt. Diese Funktion verwandelt eine Zahl in das nach dem ASCII-Code zu dieser Zahl gehörende Zeichen. Der Befehl

```
PRINT CHR$(77)
```

etwa würde auf dem Bildschirm den Buchstaben „M" ausgeben.

Übung 3

Entwerfen Sie ein BASIC-Programm, das auf dem Bildschirm die zu den ASCII-Werten 33 bis 126 gehörenden Zeichen zusammen mit diesen Werten ausgibt. Der Ausdruck auf dem Bildschirm sollte folgendermaßen aussehen:

```
ASCII-WERT 33 entspricht dem Zeichen !
ASCII-WERT 34 entspricht dem Zeichen " usw.
```

Wie oben gesagt bedient sich die dargestellte Prüfroutine des kleinen Kunstgriffs, die eingegebene Zahl zunächst als String zu behandeln, um für den Fall einer fehlerhaften Eingabe die BASIC-Fehlermeldung „?Redo from start" abzufangen. Das hat aber nun zur Folge, daß das Programm im weiteren Verlauf die so dargestellte Ziffer nicht als Ziffer erkennen kann, denn sie ist ihm ja nur als nicht näher spezifiziertes Zeichen vorgestellt worden. Man muß deshalb, nachdem der Prüfteil des Programms abgeschlossen ist, den Inhalt von ZAHL$ in eine Zahl zurückverwandeln. Dies geschieht mit Hilfe der Funktion VAL. Die folgende Zeile würde, zu dem obigen Programmstück hinzugefügt, den Inhalt von ZAHL$ als Ziffer der numerischen Variablen ZAHL zuweisen:

```
45 ZAHL=VAL(ZAHL$)
```

Mit der numerischen Variablen ZAHL kann man dann im weiteren Verlauf des Programms die gewünschten Rechenoperationen ausführen.

Wie das Programmbeispiel zeigt, schließt eine auf den Benutzer Rücksicht nehmende Behandlung fehlerhafter Eingaben am Bildschirm erscheinende Texthilfen ein. Es ist nicht immer einfach, derartige Hinweise kurz und präzise abzufassen. Wenn man sich darüber im klaren ist, gewinnt der schon mehrfach angesprochene urheberrechtliche Aspekt noch eine weitere Dimension. Derartige textliche Bestandteile eines Programms müßten nach normalen urheberrechtlichen Maßstäben geschützt sein, ohne daß es insoweit eines Eingehens auf das Problem der Schutzwürdigkeit des Programms im übrigen bedürfte. Soweit ersichtlich, ist dieser Gesichtspunkt der Einbeziehung von Texten in Programme noch nicht ausreichend als urheberrechtlich relevanter Aspekt gewürdigt worden. Das ist deswegen so bedauerlich, weil der Entwurf dieser Texte bei komplexen Programmen sehr hohe analytische und gestalterische Anforderungen stellt. Das ist übrigens nicht nur für die hier betrachteten Fehlermeldungen der Fall, sondern vielleicht in noch stärkerem Maße für die Hilfs- und Erläuterungstexte, die Bestandteil der Programme sind. Um auch dazu ein Beispiel zu nennen: Das in IuR 5/1986 vorgestellte Festplattenverwaltungsprogramm Q-DOS integriert einen etwa 70.000 Buchstaben umfassenden Hilfstext in den Programmcode. (Dieser Text wird beim Kompilieren eingebunden.) Diese Hilfe ist noch dazu „kontextsensitiv" aufgebaut, was eine besonders intelligente Programmierung voraussetzt. („Kontextsensitiv" bedeutet folgendes: Der Benutzer kann an jeder Stelle des Programms die Hilfsfunktion auslösen. Das Programm ermittelt dann die Stelle, an der die Unklarheit aufgetreten ist, und präsentiert auf dem Bildschirm die für diesen „Kontext" relevante Hilfsinformation.) Daß ein Erläuterungsbuch dieser Art urheberrechtlich geschützt wäre, unterliegt keinem Zweifel. Gleiches muß dann aber auch für den so in das Programm integrierten Text gelten, ohne daß es einer näheren Analyse der „Rechtsnatur" von Programmen bedarf.

(Mit dem letzten Gedanken soll nicht einer isolierten Betrachtung einzelner Programmkomponenten das Wort geredet werden. Gerade das Beispiel der „kontextsensitiven" Hilfsfunktion zeigt ja, wie eng Text und Programmlogik verzahnt sein müssen, damit sich das erwünschte Zusammenwirken einstellt. Es soll nur gezeigt werden, daß schon eine isolierte Betrachtung der

in Programmen enthaltenen Textkomponenten auf dem Boden des traditionellen Urheberrechts neue Schutzaspekte eröffnet. Das gleiche gilt dann natürlich auch für Bild- und Tonelemente, die wie Texte Programmteile sein können und es in immer stärkerem Maße auch sind.)

*3.1.2.1.1 Eingabe einer Ziffer statt eines Buchstabens*

Mit dem Instrumentarium der ASCII-Werte kann man jetzt auch den Fehlerfall behandeln, in dem nicht (wie eben) ein Buchstabe statt der erwarteten Ziffer eingegeben wird, sondern eine Ziffer anstelle von Buchstaben. Wie das folgende kleine Programmstück zeigt, bedarf es dazu im wesentlichen nur einer Veränderung der Prüfbedingung:

```
30 INPUT „Bitte geben Sie einen Buchstaben ein ! “, ZAHL$
40 IF ASC(ZAHL$)⟩47 AND ASC(ZAHL$)⟨58 THEN GOSUB 100:GOTO 30
50 STOP
100 REM Text fuer den Eingabefehler: Zahl statt anderem Zeichen
110 PRINT „Es muss ein Buchstabe eingegeben werden.“
120 PRINT „Sie haben “;ZAHL$; „eingegeben.“
130 RETURN
```

Ein Lauf mit fehlerhafter Eingabe sieht hier folgendermaßen aus:

```
Bitte geben Sie einen Buchstaben ein ! ⟨6⟩⟨RETURN⟩
Es muss ein Buchstabe eingegeben werden.
Sie haben 6 eingegeben.
Bitte geben sie einen Buchstaben ein !
```

Zu Zeile 40, die auch hier den Programmablauf steuert, genügt nach den Erläuterungen im vorigen Punkt der Hinweis, daß Zeichen mit ASCII-Werten über 47 und unter 58 gerade die Ziffern sind.

Auflösungen der Übungen aus der vorherigen Folge

Übung 1

a. Formel aus Schritt 2 in Übersicht 5 in BASIC-Schreibweise:

y = 1/((x-x)/x)

b. Formel aus Schritt 3 in Übersicht 5 in BASIC-Schreibweise:

y = 1/(1/x)

(Die Antwort war unter 2.3.2.5 bereits in der vorigen Folge enthalten.)

Übung 2

Ein Programm, das den gestellten Anforderungen genügt, könnte folgendermaßen aussehen:

```
10 REM Programmname: UEBUNG2.BAS
20 INPUT „Bitte geben Sie eine Zahl ein ! “,ZAHL
30 ZWISCHENERGEBNIS = ZAHL-1
40 PRINT „Ergebnis fuer (ZAHL-1) = “;ZWI
SCHENERGEBNIS
50 ZWISCHENERGEBNIS = ZWISCHENERGEB
NIS/ZAHL
60 PRINT „Ergebnis fuer ((ZAHL-1)/ZAHL) = “;
ZWISCHENERGEBNIS
70 ZWISCHENERGEBNIS = 1-ZWISCHENER
GEBNIS
80 PRINT „Ergebnis fuer (1-(ZAHL-1)/ZAHL) = “;
ZWISCHENERGEBNIS
90 ZWISCHENERGEBNIS = 1/ZWISCHENER
GEBNIS
100 PRINT „Ergebnis fuer (1/1-(ZAHL-1)/ZAHL) = “;
ZWISCHENERGEBNIS
110 STOP
```

Dieses Programm liefert bei Eingabe von „8“ folgende Ausgabe auf dem Bildschirm:

```
Ergebnis fuer (ZAHL-1) = 7
Ergebnis fuer ((ZAHL-1)/ZAHL) = .875
Ergebnis fuer (1-(ZAHL-1)/ZAHL) = .125
Ergebnis fuer (1/1-(ZAHL-1)/ZAHL) = 8
```

Bei Eingabe von „88“ ergibt sieh folgendes Bild:

```
Ergebnis fuer (ZAHL-1) = 87
Ergebnis fuer ((ZAHL-1)/ZAHL) = .9886364
Ergebnis fuer (1-(ZAHL-1)/ZAHL) = 1.136363E-02
Ergebnis fuer (1/1-(ZAHL-1)/ZAHL) = 88.00008
```

An Hand dieser Zwischenergebnisse läßt sich Schritt für Schritt nachvollziehen, wie es zu den in der vorigen Folge (vgl. 2.3.2.4) beschriebenen Fehlerbedingungen kommt.

Zum Schluß noch eine Anmerkung. Es irritiert in dem obigen Programm auf den ersten Blick, daß die Variable ZWISCHENERGEBNIS so häufig vorkommt. Intuitiv würde man erwarten, daß das erste Zwischenergebnis der Variablen ZWISCHENERGEBNIS1 zugewiesen wird, das zweite Zwischenergebnis der Variablen ZWISCHENERGEBNIS2 usw. Das wäre auch möglich. Es würde aber gegen das Prinzip der sparsamen Verwendung von Variablen verstoßen, das zwei Zwecke hat: Es dient erstens der Übersichtlichkeit von Programmen und verringert zweitens den Speicherbedarf. Man sollte sich deswegen immer wieder vor Augen führen, daß Variable wie aus dem Programm UEBUNG2.BAS ersichtlich verwendet werden können (vgl. dazu nochmals die Erläuterungen in IuR 4/86, 2.2.1, S. 183). Etwas anderes würde lediglich dann gelten, wenn im weiteren Verlauf des Programms die einzelnen Zwischenergebnisse gesondert von Interesse wären. Dann müßte man unterschiedliche Variablennamen für sie einführen.

(wird fortgesetzt)