

+++ ADOK SOFT PRESENTS +++ ADOK SOFT PRESENTS +++ ADOK SOFT PRESENTS +++

THE REAL ADOK'S WAY TO

# QBASIC

A NEW METHOD STEP BY STEP  
WRITTEN BY CLAUDIUS-DIETER VOLKO, VIENNA

## INHALTSVERZEICHNIS

Kapitel 1.....	Erste Schritte
Kapitel 2.....	Grundlegende Befehle zur Verarbeitung von Zahlen
Kapitel 3.....	Einfache grafische und musikalische Effekte
Kapitel 4.....	Debugging in Quick Basic und Power Basic
Kapitel 5.....	Blöcke
Kapitel 6.....	Programmierung von Abenteuerspielen
Kapitel 7.....	Grafikprogrammierung
Kapitel 8.....	Datentypen in QBasic
Kapitel 9.....	Dateiverwaltung
Kapitel 1.....	Befehle und Funktionen
Kapitel 2.....	Funktionen zur Verarbeitung von Strings
Kapitel 3.....	Die QBasic-Toolbox
Nachwort	

Alle Beispielprogramme sind auf der beiliegenden Diskette als *BAS*-Dateien enthalten! Sie können unter QBasic geladen und sofort gestartet werden.

## KAPITEL 1

### Erste Schritte

Es gibt anscheinend ziemlich viele Leute, die das Programmieren in Basic erlernen wollen und sehnsüchtig auf einen Kurs warten. Deshalb entschloß ich mich, mit einem Kurs anzufangen.

In meinem Kurs behandeln wir den Interpreter QBasic, der bei MS-DOS ab Version 5.0 beiliegt. Falls ihr ein anderes Betriebssystem benutzt, in dem QBasic nicht enthalten ist, könnt ihr auch den noch leistungsfähigeren und 99,9%ig kompatiblen Compiler Quick Basic ab Version 4.0 verwenden. (Die weitverbreitete Meinung, Quick Basic sei 100%ig kompatibel, ist nicht ganz richtig. Wenn man *CALL ABSOLUTE* anwendet, muß man Quick Basic mit dem Parameter */L* starten, während bei QBasic der Parameter entfallen kann. Dazu aber noch später.) Falls ihr weder QBasic noch Quick Basic besitzt, könnt ihr auch den Profi-Compiler Power Basic verwenden. Ihr müßt dabei nur beachten, daß einige Befehle anders anzuwenden sind. Sollte ein solcher Befehl besprochen werden, so werde ich es im Text erwähnen.

Nun, laßt uns mit dem Kurs beginnen.

QBasic läßt sich durch Eingabe von *QBASIC* am DOS-Prompt starten. Binnen weniger Sekunden ist der Editor geladen. Es erscheint eine graue Dialogbox. Ihr könnt jetzt wählen, ob ihr zuerst eine Einführung in die Bedienung des Editors erhalten, oder euch gleich ins Programmiervergnügen stürzen wollt.

Falls ihr euch noch nicht mit der Bedienung der DOS-Programme *EDIT* und *HELP* auskennt, die im Prinzip nichts anderes als abgespeckte Versionen des QBasic-Editors und der QBasic-Online-Hilfe sind, solltet ihr euch für die Einführung in die Bedienung des Editors entscheiden.

Ihr wählt diesen Menüpunkt, indem ihr die Enter-Taste betätigt. Plötzlich wird der Bildschirm schwarz, und es erscheinen komische weiße Buchstaben. Dies ist die Online-Hilfe von QBasic. Ihr könnt mit den Cursortasten und Page-Up bzw. Page-Down in ihr herumreisen. Laßt euch Zeit und lest alles genau von oben bis unten durch.

Bald werdet ihr auf das Ende der Online-Hilfe stoßen. "DAS WAR SCHON ALLES? Die berühmte Online-Hilfe habe ich mir länger vorgestellt!" werdet ihr jetzt rufen. Halt, das war noch nicht alles! Reist mit den Cursortasten wieder zurück zum Anfang der Hilfe. Was erkennt ihr dort? Grüne Pfeile, zwischen denen sich ein kurzer Text befindet! Wozu sollen die gut sein? Probiert sie einfach aus!

Geht mit dem Cursor auf den Text, der zwischen den grünen Pfeilen steht, und drücke einmal die Enter-Taste. "Was? Das geht ja noch weiter!" ruft ihr entsetzt. Richtig, denn die Texte zwischen den grünen Pfeilen sind Menüpunkte, mit denen ihr in weitere Gebiete der riesigen Online-Hilfe vordringen könnt.

Jetzt werdet ihr euch sicherlich fragen, wozu das ganze gut sein soll. Ganz einfach: Nehmen wir an, ihr wäret gerade dabei, ein QBasic-Programm zu erstellen, und euch fällt auf einmal nicht mehr ein, wie ein bestimmter Befehl funktioniert. Dann fahrt einfach mit dem Cursor auf diesen Befehl und drückt F1. Damit wird die Online-Hilfe für diesen Befehl aufgerufen. Ihr könnt euch jetzt in Ruhe die Anweisungen durchlesen und dann mit einem Tastendruck auf Escape wieder zum Editor zurückkehren.

Wenn ihr nun mit dem Schmökern in der Online-Hilfe fertig seid und voller schöpferischer Energie euch auf das Programmieren freut, drückt die Escape-Taste. Schwuppdwupp ist der schwarze Bildschirm verschwunden. Stattdessen lächelt euch eine freundliche, blaue Oberfläche an. Sie wird euch beim Programmieren immer begleiten. Ihr läuft schon jetzt langsam die Gefahr, euch zu verlieben. Sie wartet ergeben mit einer Eingabeaufforderung auf euch. Ihr seid am Zug. Fangen wir an, ein erstes Programm zu schreiben.

Ihr könnt ein Programm genauso wie einen normalen Text eingeben. QBasic ist nämlich nichts anderes als ein erweiterter MS-DOS-Texteditor. Tippt nun genauso, wie ihr mit dem Editor einen Brief schreiben würdet, folgendes Programm ab. Der Name des Programms lautet *KURS0001*.

```
CLS
PRINT "Hallo, Gebieter! Wie soll ich dich ansprechen?"
PRINT "Gib deinen Namen ein!"
LINE INPUT name$
PRINT "Hallo, ";
PRINT name$;
PRINT ", ich will all deine Wünsche erfüllen."
PRINT "Ich will dir jetzt zeigen, mit welchen Befehlen du mich"
PRINT "beherrschen kannst."
```

Wenn ihr dieses Programm fertig abgetippt habt, könnt ihr es durch das Drücken der Taste F5 starten.

Zuerst wird der Bildschirm gelöscht. Danach erscheint folgender Text: *Hallo, Gebieter! Wie soll ich dich ansprechen? Gib deinen Namen ein!*

Jetzt seid ihr an der Reihe. Gebt entweder euren Namen, das Pseudo oder was auch immer ein. Mit Enter schließt ihr die Eingabe wieder ab.

Jetzt erscheint ein weiterer Text: *Ich will dir jetzt zeigen, mit welchen Befehlen du mich beherrschen kannst.* Danach erscheint die Meldung *"Beliebige Taste drücken, um fortzusetzen"*. Nach einem weiteren Tastendruck wird das Programm beendet.

Das war also unser erstes Programm, das wir erstellt haben! So einfach geht das. Super, nicht wahr? Aber wie funktioniert das Programm eigentlich?

Die erste Zeile des Programms lautet *CLS*. *CLS* ist ein Befehl, welcher QBasic befiehlt, den Bildschirm schwarz zu machen, also den vorherigen Inhalt zu löschen.

Wenn ein Programm gestartet wird und QBasic auf einen Befehl trifft, übersetzt es ihn in eine bestimmte Reihe von Impulsen, die der Computer versteht.

Die zweite Zeile beginnt mit *PRINT*. Dies ist ebenfalls ein Befehl. Im Gegensatz zu *CLS* hat *PRINT* mehrere Funktionen, weil er mit Parametern aufgerufen werden kann. *"Hallo, Gebieter! Wie soll ich dich ansprechen?"* ist zum Beispiel ein solcher Parameter.

*PRINT* dient dazu, Zeichenketten oder Zahlen auf dem Bildschirm auszugeben. In unserem Beispiel gibt *PRINT* folgendes aus: *Hallo, Gebieter! Wie soll ich dich ansprechen?* Dies ist eine Zeichenkette. Zeichenketten müssen in QBasic immer in Anführungszeichen eingeschlossen sein. Ansonsten würde der Interpreter diesen Text als Variablen interpretieren.

Die dritte Zeile verstehen wir jetzt schon. Kommen wir also zur vierten Zeile. Hier finden wir den Befehl *LINE INPUT*. Was ist das schon wieder? Mit *LINE INPUT* lassen sich Zeichenketten von der Tastatur eingeben. In diesem Fall fragte der Computer in der dritten Zeile nach eurem Namen. Der Satz bleibt auf dem Bildschirm stehen, und mit *LINE INPUT* erwartet der Computer die Eingabe eures Namens.

Als Parameter von *LINE INPUT* steht im Beispielprogramm *name\$*. Das Dollarzeichen bezeichnet man als *String*. Man erhält es, indem man gleichzeitig Umschalt und 4 betätigt.

Das Stringzeichen gibt an, daß *name\$* ein Platzhalter für Zeichenkettenvariablen ist. Nachdem man den gewünschten Namen eingegeben und Enter gedrückt hat, wird dieser Name im Platzhalter *name\$* abgespeichert. Jetzt können alle anderen Befehle auf diesen ausgefüllten Platzhalter zurückgreifen.

Man nennt einen solchen Platzhalter eine Stringvariable. Wir werden später noch andere Variablen kennenlernen. Es klingt am Anfang ein bißchen kompliziert, ist aber sehr einfach.

Die fünfte Zeile ist wieder ein *PRINT*-Befehl. Nach dem Parameter *"Hallo, "* steht allerdings ein Punktstrich (Semikolon). Wie kluge Leser sicherlich gemerkt haben, dient er dazu, daß der nächste Text oder die nächste Zahl direkt neben diesem erscheint.

In unserem Programm wird zuerst die Programmzeile fünf ausgeführt und der Text *'Hallo, '* auf dem Bildschirm ausgegeben.

Zeile sechs ist wieder eine Bildschirmausgabe, und weil am Ende der Zeile fünf ein Punktstrich stand, wird sie direkt neben dem Text der Zeile fünf auf dem Bildschirm geschrieben.

Die Zeile sechs beinhaltet noch eine Besonderheit: Als Parameter wird diesmal kein in Anführungszeichen eingeschlossener Text, sondern die Stringvariable *name\$* angegeben. Das bedeutet, daß kein vorgegebener Text, sondern der Inhalt der Variablen ausgegeben wird.

Neben *name\$* steht wieder ein Punktstrich, wodurch, wie wir bereits oben erwähnt haben, der Zeilenvorschub unterdrückt und der Text, der in der siebten Programmzeile angegeben ist, in derselben Zeile ausgegeben wird.

Am Ende der siebten Programmzeile steht diesmal kein Punktstrich, wodurch der Text der achten Programmzeile wieder in einer eigenen Zeile ausgegeben wird.

Fertig! Das war das ganze Programm. Jetzt können wir uns wieder entspannt zurücklehnen.

Ich möchte noch einige Tips zum weiteren Verlauf dieses Kurses geben. Man sollte die Beispielprogramme immer ausdrucken und das Listing vor den Augen haben. Alle Anweisungen in diesem Text beziehen sich auf die Listings. Die Beispielprogramme kann man mit dem Menüpunkt *Datei-Drucken* ausdrucken.

Das eingetippte Programm kann man mit dem Menüpunkt *Datei-Speichern* abspeichern. Wenn ihr das Programm zum ersten Mal abspeichert, werdet ihr nach dem Dateinamen gefragt, unter dem es gespeichert werden soll. Für dieses Programm schreibt *KURS0001*. QBasic ergänzt automatisch die Dateierweiterung *BAS*. Der komplette Name wird nach dem Abspeichern *KURS0001.BAS* heißen.

Nach dem Ende einer Lektion solltet ihr mit den erlernten Befehlen experimentieren. Dazu gebe ich 'Hausaufgaben', die ihr lösen sollt. Die richtigen Ergebnisse findet ihr im Anhang.

**Aufgabe 1:** Entwerft ein Programm, das zuerst den Benutzer nach seinem Namen und danach nach seinem Pseudo fragt. Anschließend soll der Computer folgenden Text auf dem Bildschirm ausgeben: *"Guten Tag, <name>! Dein Pseudo lautet <pseudo>. Willkommen im Club der QBasic-Programmierer!"* *<name>* steht dabei für die Variable, in der der Name, und *<pseudo>* für die Variable, in der das Pseudo abgespeichert ist. In meinem konkreten Fall würde der Satz lauten: *"Guten Tag, Claus-Dieter Volko! Dein Pseudo lautet The Real Adok. Willkommen im Club der QBasic-Programmierer!"* Beachtet, daß ihr abweichend vom Programmbeispiel *KURS0001* diesmal zwei Stringvariablen verwenden müßt.

**Aufgabe 2:** Nun sollt ihr ein Programm schreiben, das den Namen des Benutzers und den seiner Freundin einliest. Anschließend soll es einen lustigen Satz über die beiden auf dem Bildschirm ausgeben, etwa so: *"<freundin>: 'Wann wirst du mich endlich heiraten, <benutzer>?'"*

**Aufgabe 3:** Jetzt etwas für die Fortgeschrittenen. Es gibt einen Befehl namens *LPRINT*, mit dem man Texte auf dem Drucker ausgeben kann. Er funktioniert ansonsten wie *PRINT*. Entwerft ein Programm, das zunächst eine oder mehrere Zeichenketten vom Benutzer eingeben läßt und sie anschließend zusammen mit einem anderen Text auf dem Drucker ausgibt. Falls ihr Probleme mit *LPRINT* haben solltet, schaut in der Online-Hilfe nach!

## Lösungen zu Kapitel 1

**Aufgabe 1:** Hier müßt ihr ein Programm schreiben, das zuerst den Benutzer nach seinem Namen und seinem Pseudo fragt und anschließend einen Text ausgibt.

Eine mögliche Lösung wäre das folgende Listing:

```
CLS
PRINT "Wie lautet dein Name?"
LINE INPUT name$
PRINT "Und wie lautet dein Pseudo?"
LINE INPUT pseudo$
PRINT "Guten Tag, " ;
```

```
PRINT name$;
PRINT "! Dein Pseudo lautet ";
PRINT pseudo$;
PRINT "."
PRINT "Willkommen im Club der QBasic-Programmierer!"
```

Zur Wiederholung: Mit der ersten Zeile wird der Bildschirm gelöscht. Man sollte am Anfang eines jeden Programms den Bildschirm löschen, weil es einfach schöner und übersichtlicher aussieht.

Mit der zweiten Zeile wird der Text *Wie lautet dein Name?* auf dem Bildschirm ausgegeben.

In der dritten Zeile steht die Anweisung, mit der der Name eingegeben und in der Variablen *name\$* gespeichert wird.

Vierte und fünfte Zeile funktionieren genauso, nur wird diesmal das Pseudo eingegeben.

Mit den restlichen sechs Zeilen wird der Text auf dem Bildschirm ausgegeben. Zur Erinnerung: Der Punktstrich bzw. das Semikolon verhindert einen Zeilenvorschub. Das heißt, daß der nächste Text gleich anschließend und nicht erst in der nächsten Zeile ausgegeben wird.

**Aufgabe 2:** Ihr müßtet ein Programm schreiben, das den Namen des Benutzers und den seiner Freundin einliest. Anschließend soll ein lustiger Text ausgegeben werden.

Hier eine mögliche Lösung:

```
CLS
PRINT "Wie heißt du?"
LINE INPUT benutzer$
PRINT "Und wie heißt deine Freundin bzw. dein Freund?"
LINE INPUT freundin$
PRINT freundin$; ": 'Wann wirst du mich endlich heiraten, "; benutzer$; "?'"
```

Ihr seht, im Prinzip funktioniert es genauso wie in der ersten Aufgabe. Nur die letzte Zeile sieht etwas komisch aus. Keine Panik, es ist nur eine verkürzte Form mehrerer *PRINT*-Befehle. Statt der letzten Zeile könnte man auch

```
PRINT freundin$;
PRINT ": 'Wann wirst du mich endlich heiraten, ";
PRINT benutzer$;
PRINT "?'"
```

schreiben. Solange es nicht auf Kosten der Übersicht geht, würde ich aber lieber zur verkürzten Methode raten.

**Aufgabe 3:** Die dritte und letzte Hau-Sauf-Gabe ;-) war ein Programm, das mehrere Zeilen vom Benutzer einliest und sie mit Hilfe des Befehls *LPRINT* auf dem Drucker ausgibt.

Hier eine mögliche Lösung:

```
CLS
PRINT "Bitte gib drei kurze Sätze ein!"
LINE INPUT satz1$
LINE INPUT satz2$
LINE INPUT satz3$
LPRINT "Der 1. Satz lautet:"
LPRINT satz1$
LPRINT "Der 2. Satz lautet:"
LPRINT satz2$
LPRINT "Der 3. Satz lautet:"
LPRINT satz3$
```

Dieses Listing könnte man noch optimieren, aber dazu später, wenn wir über die *Arrayverwaltung* und die *FOR/NEXT*-Schleife sprechen werden.

## KAPITEL 2

### Grundlegende Befehle zur Verarbeitung von Zahlen

Das letzte Mal lernten wir die Befehle *CLS*, *PRINT* und *LINE INPUT* kennen. *PRINT* und *LINE INPUT* sind die grundlegenden Befehle zur Verarbeitung von Zeichenketten (Strings). Nun wollen wir zur Abwechslung die Zahlenvariablen, wie wir sie aus der Mathematik kennen, unter die Lupe nehmen.

Dazu gleich ein Beispielprogramm. Gebt es in QBasic ein und speichert es unter *KURS0002.BAS*.

```
CLS
zahl1 = 12
zahl2 = 3
PRINT "Zahl 1:";
PRINT zahl1
PRINT "Zahl 2:";
PRINT zahl2
PRINT "Addition:";
PRINT zahl1 + zahl2
PRINT "Subtraktion:";
PRINT zahl1 - zahl2
PRINT "Multiplikation:";
PRINT zahl1 * zahl2
PRINT "Division:";
PRINT zahl1 / zahl2
```

Schon haben wir ein einfaches Rechenprogramm! Startet es, und QBasic führt euch alle Grundrechnungsarten vor.

Wenn ihr andere Zahlen als 12 und 3 einsetzen wollt, so müßt ihr einfach die entsprechenden Werte in die Zeilen 2 und 3 schreiben.

*zahl1* und *zahl2* sind, wie sich manche von euch wahrscheinlich schon denken werden, Zahlenvariablen. Sie sind Platzhalter für beliebige Fließkommawerte, die eine bestimmte Größe und Anzahl von Nachkommastellen haben können.

Es gibt noch andere sogenannte *Datentypen*, in denen man Zahlen speichern kann. Darauf möchte ich erst später eingehen. Im Moment reicht es, zu wissen, daß man in Variablen, die kein Stringzeichen (\$) enthalten, Fließkommawerte speichern kann.

Wie das Programm zeigt, lassen sich mit *PRINT* auch Zahlenwerte auf dem Bildschirm ausgeben. Vor und nach der Zahl wird jeweils ein Leerzeichen ausgegeben. Bei negativen Zahlen entfällt das erste Leerzeichen. Stattdessen wird das Vorzeichen ausgegeben. Ihr könnt dies leicht überprüfen, indem ihr in der Zeile 2 oder 3 für eine der beiden Variablen einen negativen Wert einsetzt.

Die Zeichen '+', '-', '\*' und '/' sind, wie ihr sicherlich schon bemerkt habt, Rechenzeichen, wobei '/' für Division steht. Dieses Rechenzeichen sollte man sich besonders gut merken, weil es von der schriftlichen Norm abweicht.

Anfänger könnten auch leicht '\*' (Multiplikationszeichen) mit ',' (Komma) verwechseln. Die englische Schreibweise des Kommas und der Rechenzeichen wird übrigens in den meisten Anwendungsprogrammen und Programmiersprachen verlangt. Basic ist daher kein Einzelfall, und falls ihr doch nicht mehr weiterprogrammieren wollt (zu schwierig!?), solltet ihr euch diese Schreibweise trotzdem merken.

Natürlich kann man nicht nur die Ergebnisse der Rechnungen anzeigen, sondern sie auch anderen Variablen zuweisen. Hier ein weiteres Beispielprogramm. Speichert es unter *KURS0003.BAS* und startet es mit *F5*.

```
CLS
zahl1 = 100
DO
  zahl1 = zahl1 + 1.6
  PRINT zahl1
LOOP
```

Das Programm läuft unendlich lange. Keine Panik, der Computer ist nicht abgestürzt! Durch gleichzeitiges Betätigen von *STRG* und *PAUSE* läßt sich das Programm unterbrechen.

Ihr gelangt in QBasic zurück. Die Programmzeile, bei deren Ausführung ihr *STRG* und *PAUSE* gedrückt habt, ist hervorgehoben. Kümmert euch jedoch nicht darum.

Nun möchte ich das Programm Zeile für Zeile beschreiben. Zeile 1 löscht den Bildschirm, Zeile 2 setzt die Variable *zahl1* auf 100.

Das habe ich vorhin vergessen zu erwähnen: Das Gleichheitszeichen kann in Basic verschiedene Bedeutungen haben. Hier bedeutet es *Zuweisung*. Das heißt, daß der Variablen, die links von dem Gleichheitszeichen steht, der Wert rechts von dem Gleichheitszeichen zugewiesen wird. Eine Anweisung wie

```
100 = zahl1
```

führt zu einer Fehlermeldung.

Man kann übrigens auch *LET* vor die Variable schreiben. Unser Beispiel würde dann folgendermaßen aussehen:

```
LET zahl1 = 100
```

Die Verwendung von *LET* ist aber heutzutage weder notwendig noch empfehlenswert.

Die nächste Zeile enthält ein neues Schlüsselwort. Es handelt sich um *DO*, einen sehr flexiblen und vielfältig einsetzbaren Befehl, den wir im Kapitel über Schleifen und Verzweigungen genau besprechen werden. Jetzt sei nur gesagt, daß der Programmteil zwischen *DO* und *LOOP* unendlich oft ausgeführt wird.

In der 4. Zeile wird der Variablen *zahl1* das Ergebnis der Addition  $zahl1 + 1.6$  zugewiesen. Das ist einfacher, als es klingt, denn es bedeutet nichts anderes, als daß *zahl1* um 1,6 erhöht wird.

Danach wird die Zahl auf dem Bildschirm ausgegeben, und mit *LOOP* wird das ganze ab der vierten Zeile wiederholt. Alles klar?

**Aufgabe 1:** Zurück zum ersten Beispielprogramm dieses Kapitels. Wie wäre es, wenn der Benutzer die Zahlen selbst eingeben könnte? Dazu gibt es den Befehl *INPUT*. Er funktioniert genauso wie *LINE INPUT*, nur wird zusätzlich ein Fragezeichen auf dem Bildschirm ausgegeben. Man kann zwar mit *INPUT* auch Zeichenketten eingeben, doch dazu eignet sich *LINE INPUT* meiner Meinung nach besser, weil sich damit auch Beistriche eingeben lassen. Erstellt als Hausaufgabe ein Rechenprogramm, in dem der Benutzer die Zahlen selbst eingeben kann!

## Lösungen zu Kapitel 2

**Aufgabe 1:** Ihr müßtet unter Verwendung des *INPUT*-Befehls das Programm *KURS0002.BAS* so verändern, daß der Benutzer die Zahlen selbst eingeben kann. Hier eine mögliche Lösung:

```
CLS
PRINT "Wie lautet die erste Zahl";
INPUT zahl1
PRINT "Wie lautet die zweite Zahl";
INPUT zahl2
CLS
PRINT "Zahl 1:";
PRINT zahl1
PRINT "Zahl 2:";
PRINT zahl2
PRINT "Addition:";
PRINT zahl1 + zahl2
PRINT "Subtraktion:";
PRINT zahl1 - zahl2
PRINT "Multiplikation:";
```

```
PRINT zahl1 * zahl2
PRINT "Division:";
PRINT zahl1 / zahl2
```

Ihr seht, daß der Befehl *INPUT* fast genauso wie *LINE INPUT* funktioniert.

Wenn *INPUT* ausgeführt wird, erscheint auf dem Bildschirm ein Fragezeichen. Das Programm verlangt nun, daß der Benutzer einen Text oder eine Zahl eingibt. Die Eingabe wird mit der *ENTER*-Taste bestätigt, und der eingegebene Text bzw. die eingegebene Zahl wird in der Variablen gespeichert, die im Listing neben *INPUT* steht.

Die Unterschiede zwischen *INPUT* und *LINE INPUT* bestehen also darin, daß *INPUT* ein Fragezeichen ausgibt und der Benutzer nicht nur Zeichenketten, sondern auch Zahlen eingeben kann. (Um Mißverständnissen vorzubeugen: Zeichenketten können auch Ziffern enthalten!)

Wenn ihr das obenstehende Programm gestartet habt, werdet ihr bemerkt haben, daß das Fragezeichen, das *INPUT* auf dem Bildschirm ausgibt, direkt neben den Ausgaben der *PRINT*-Anweisungen erscheint. Das heißt im Klartext, daß sich das Semikolon nicht nur auf die nächste *PRINT*-, sondern überhaupt auf die nächste Anweisung, die eine Bildschirmausgabe bewirkt, bezieht.

Auch *LINE INPUT* bewirkt eine Bildschirmausgabe: Der Cursor und die eingegebenen Zeichen werden auf dem Bildschirm ausgegeben!

## KAPITEL 3

### Einfache grafische und musikalische Effekte

Wie kann man einfache Grafik- und Soundeffekte realisieren, um die eigenen Programme aufzuwerten? Dazu gibt es mehrere Befehle, die ich jetzt aufzählen und näher erklären möchte.

Es ist vielleicht nicht so schön, wenn bei einem Programm die Textausgaben nacheinander folgen. Deshalb gibt es den Befehl *LOCATE*. Mit ihm kann man den Cursor an einer beliebigen Stelle des Bildschirms plazieren. Die "Syntax" des Befehls, das heißt, seine Schreibweise, lautet:

```
LOCATE zeile, spalte
```

Diese Syntax verwenden übrigens alle Basic-Dialekte auf dem PC. Auf den meisten anderen Computern wird zuerst die Spalte angegeben.

Der Textmodus, auf den wir uns (vorläufig) beschränken, hat eine Auflösung von 80 Spalten mal 25 Zeilen. Mit dem *LOCATE*-Befehl positionieren wir quasi den Cursor auf die durch die Variablen *zeile* und *spalte* angegebene Position. Die nächste Bildschirmausgabe findet an dieser Position statt, was sehr praktisch ist.

Man kann auch wahlweise *zeile* oder *spalte* weglassen. Der nicht angegebene Wert wird dann automatisch eingefügt.

Beispiel: Wenn wir uns in Zeile 12, Spalte 10 befinden und den Befehl

```
LOCATE , 40
```

verwenden, wird der Cursor an die Position Zeile 12, Spalte 40 gesetzt. Hier wurde also die Zeile nicht angegeben, dafür aber die aktuelle Zeile automatisch eingesetzt.

Um die Fähigkeiten des *LOCATE*-Befehls zu demonstrieren, hier eine erweiterte Version unserer Hausaufgabe:

```
CLS
LOCATE 2, 2
PRINT "Wie lautet die erste Zahl";
```

```

INPUT zahl1
LOCATE 5, 2
PRINT "Wie lautet die zweite Zahl";
INPUT zahl2
CLS
LOCATE 2, 2
PRINT "Zahl 1:";
PRINT zahl1;
LOCATE , 60
PRINT "Zahl 2:";
PRINT zahl2
LOCATE 3, 1
PRINT "-----";
PRINT "-----"
LOCATE 5, 2
PRINT "Addition:";
LOCATE , 17
PRINT zahl1 + zahl2
LOCATE 6, 2
PRINT "Subtraktion:";
LOCATE , 17
PRINT zahl1 - zahl2
LOCATE 7, 2
PRINT "Multiplikation:";
LOCATE , 17
PRINT zahl1 * zahl2
LOCATE 8, 2
PRINT "Division:";
LOCATE , 17
PRINT zahl1 / zahl2

```

Damit ihr keine Tippfehler macht: In der Zeile 16 befinden sich 68 Minusse, in der Zeile 17 zehn Minusse.

Man könnte zwar auch die insgesamt 78 Minuszeichen in einer *PRINT*-Anweisung zusammenfassen, jedoch ist die Zeile dann so lang, daß sie über den Bildschirmrand hinaus geht.

Dieses Programm werden wir noch öfters verändern. Speichert es also unter *KURS0004.BAS* ab.

Bisher hatten alle Programme eine einheitliche Farbkombination: weißer Text auf schwarzem Hintergrund.

Da dies mit der Zeit sehr eintönig werden kann, gibt es in QBasic einen Befehl, um eine andere Farbkombination einzustellen: *COLOR*. Seine Syntax lautet:

```
COLOR vordergrundfarbe, hintergrundfarbe
```

Die Vordergrundfarbe ist hierbei die Farbe, in der der Text geschrieben werden soll, während die Hintergrundfarbe, wie der Name schon sagt, die Farbe des Hintergrunds ist. Wie bei *LOCATE* kann auch ein Parameter weggelassen werden.

Wie gibt man nun die Vordergrundfarbe und die Hintergrundfarbe an? Das ist leider nicht ganz so einfach.

Zu jeder Farbe gehört ein sogenannter Farbcode. Dieser Farbcode ist eine Zahl von 0 bis einschließlich 31.

Jede Zahl von 0 bis einschließlich 7 steht für eine "dunkle" Farbe. Zählt man zu einer dieser Zahlen 8 dazu, so erhält man die dazugehörige "helle" Farbe. Beispielsweise ist der Farbcode der Farbe Dunkelblau 1. Daher ist der Farbcode der Farbe Hellblau 9. Nur die Farbe mit der Nummer 8 ist hier eine Ausnahme, denn diese ist Grau.

Zählt man zu einer der Farben von 0 bis einschließlich 15 nun 16 hinzu, erhält man eine "blinkende" Farbe.

Man kommt nicht darum, die Farbcodes auswendig zu lernen. Da ich euch aber jetzt die "Philosophie" hinter der Reihenfolge der Farbcodes erklärt habe, wird es euch sicherlich leicht fallen. Hier nun die vollständige Tabelle der Farbcodes:

0 Schwarz	8 Grau	16 Blink-Schwarz	24 Blink-Grau
1 Dunkelblau	9 Hellblau	17 Blink-D.-Blau	25 Blink-H.-Blau

2 Dunkelgrün	10 Hellgrün	18 Blink-D.-Grün	26 Blink-H.-Grün
3 Dunkelzyan	11 Hellzyan	19 Blink-D.-Zyan	27 Blink-H.-Zyan
4 Dunkelrot	12 Hellrot	20 Blink-D.-Rot	28 Blink-H.-Rot
5 Dunkelviolett	13 Hellviolett	21 Blink-D.-Violett	29 Blink-H.-Violett
6 Braun	14 Gelb	22 Blink-Braun	30 Blink-Gelb
7 Weiß	15 Leucht-Weiß	23 Blink-Weiß	31 Blink-Leucht-Weiß

Wenn man eine Farbe öfters verwendet, könnte man sie auch in einer Variablen speichern. Wird z.B. die Farbe Dunkelgrün öfters benötigt, so könnte man ihren Farbcode der Variablen *gruen* zuweisen. Statt *COLOR 2* braucht man dann nur mehr *COLOR gruen* zu schreiben. QBasic ersetzt während der Ausführung automatisch den Platzhalter *gruen* durch den Wert 2.

Man könnte auch einer Variablen *hell* den Wert 8 und einer Variable *blink* den Wert 16 zuweisen. Will man dann die Vordergrundfarbe auf Blink-Hellgrün umschalten, schreibt man einfach *COLOR blink + hell + gruen*. Ihr seht, das ist unserer Umgangssprache schon sehr ähnlich!

Doch nun zu unserem Beispielprogramm. Es ist eine Weiterentwicklung von *KURS0004.BAS*, und deshalb speichern wir es *unter KURS0005.BAS*.

```
blau = 1
gelb = 14
weiss = 7
hell = 8

COLOR , blau
CLS

COLOR hell + weiss
LOCATE 2, 2
PRINT "Wie lautet die erste Zahl";
COLOR gelb
INPUT zahl1

COLOR hell + weiss
LOCATE 5, 2
PRINT "Wie lautet die zweite Zahl";
COLOR gelb
INPUT zahl2

CLS
COLOR hell + weiss
LOCATE 2, 2
PRINT "Zahl 1:";
COLOR gelb
PRINT zahl1;

COLOR hell + weiss
LOCATE , 60
PRINT "Zahl 2:";
COLOR gelb
PRINT zahl2

COLOR hell + weiss
LOCATE 3, 1
PRINT "-----";
PRINT "-----"

COLOR hell + weiss
LOCATE 5, 2
PRINT "Addition:";
COLOR gelb
LOCATE , 17
PRINT zahl1 + zahl2

COLOR hell + weiss
LOCATE 6, 2
PRINT "Subtraktion:";
COLOR gelb
LOCATE , 17
PRINT zahl1 - zahl2
```

```

COLOR hell + weiss
LOCATE 7, 2
PRINT "Multiplikation:";
COLOR gelb
LOCATE , 17
PRINT zahl1 * zahl2

```

```

COLOR hell + weiss
LOCATE 8, 2
PRINT "Division:";
COLOR gelb
LOCATE , 17
PRINT zahl1 / zahl2

```

In den Zeilen 6 und 7 wird gezeigt, daß *CLS* den Bildschirm nicht immer schwarz macht, sondern ihn in der aktuellen Hintergrundfarbe ausfüllt.

Außerdem seht ihr, daß ich in diesem Programm einige Leerzeilen eingefügt habe. Das dient schlicht und einfach dazu, daß das Programm übersichtlicher wird.

Zur Übersichtlichkeit ist es auch nützlich, wenn man den Befehl *REM* einsetzt. Alles, was nach diesem Wort in derselben Zeile steht, wird ignoriert. Das dient vor allem zur Erläuterung einzelner Programmabschnitte oder zum Ausprobieren von verschiedenen Varianten (am Anfang der Programmzeile einfach *REM* schreiben - Folge: Befehle, die sich in der Zeile befinden, werden ignoriert).

Der Befehl *REM* kann durch ein Hochkomma abgekürzt werden. Dann kann man ihn auch mitten in die Zeile hineinsetzen. In diesem Fall wird nur ab dieser Stelle die Ausführung unterbunden.

Ein kleines Beispiel (*KURS0006.BAS*):

```

'Variablenzuweisungen
blau = 1                'Farbe Blau
gelb = 14               'Farbe Gelb

'Hauptprogramm
COLOR gelb, blau       'Farben wählen
CLS                    'Bildschirm löschen
PRINT "Hallo, Leute!"  'Text ausgeben
REM PRINT "Mieses Wetter heute!" 'Ausgeklammerter Befehl
PRINT "Schönes Wetter heute!" 'Text ausgeben
PRINT "Bis morgen!"   'Text ausgeben

```

Alles klar?

*PRINT* läßt sich übrigens auch durch das Fragezeichen (?) abkürzen. Doch da dies nicht sehr gebräuchlich ist, erwähnen wir es nur am Rande.

Nun zu der Musik. (ENDLICH!!!!) Für diesen Befehl ist es notwendig, daß ihr wenigstens etwas im Musikunterricht aufgeschnappt habt. Die Notennamen möchte ich hier nicht noch einmal erklären! Alle Unmusikalischen können aber auch etwas Musik mit den Befehlen *BEEP* und *SOUND* erzeugen (in der Online-Hilfe nachlesen!).

Kommen wir aber zum gebräuchlichsten Musikbefehl in QBasic: *PLAY*. Mit ihm kann der Programmierer "Klavier spielen". Die Noten müssen dazu aber in einer Zeichenkette vorliegen. Das Format ist zwar anfangs etwas ungewöhnlich, aber da auch der mächtige Grafikbefehl *DRAW* ein ähnliches Format verwendet, lohnt es sich, es zu lernen. Sehen wir uns dazu ein Beispiel an.

```

'I want to be in America
'Coded by The Real Adok
'Übersichtliche Version für Basic-Kurs
PLAY "t120 18 o4"
PLAY "g g g > c c c < 14 a f c"           'Tempo 120, Achtel, Oktave 4
PLAY "18 g g g > c c c 14 d < b g"        'Das Lied beginnt!
PLAY "18 b- b- b- > e- e- e- 14 d < b- f"  'Weiter geht's!
PLAY "18 e- e- e- a- a- a- 14 g > 18 e p8 < 14 c" 'Ende.

```

Klingt doch gut, oder?

Der erste PLAY-Befehl stellt das Tempo auf 120, die Notenlänge auf Achtel und die Oktave auf 4 ein.

Man erkennt: Der Buchstabe 't' steht für das Tempo, 'l' für die Notenlänge und 'o' für die Oktave. Bei der Oktave müssen Zahlen angegeben werden, denn QBasic versteht sowas wie 'große Oktave' oder 'eingestrichene Oktave' nicht. 0 ist die tiefste und 7 die höchste Oktave. Standard ist 4.

Das Tempo entspricht der Taktfrequenz des Metronoms und reicht theoretisch von 0 bis 255. Tatsache ist aber, daß ein Tempo unter 32 nicht auszuhalten ist.

Schließlich und endlich gibt die Zahl neben dem 'l', wie gesagt, die Notenlänge an. Wenn diese Zahl z.B. x ist, dann ist die Notenlänge 1/x (logisch, man will es dem Programmierer auch nicht zu kompliziert machen).

Das waren die allgemeinen Vorbereitungen. Jetzt beginnt das eigentliche Lied! Natürlich lassen sich die Befehle, die wir im Vorbereitungsteil verwendet haben, auch noch jetzt verwenden. Allerdings kenne ich kaum Lieder, in denen das Tempo ständig gewechselt wird (außer in meinen eigenen :-)) und denen von Bela Bartok)...

Noten werden durch Nennung ihres Namens gespielt. Hier haben wir beispielsweise gleich zu Anfang drei G's gespielt, die in der aktuellen Oktave im aktuellen Tempo und der aktuellen Notenlänge ertönen. Wichtig ist nur, daß die englische Schreibweise benutzt wird, also überall dort, wo die Note H vorkommt, B geschrieben wird. Das ist die einzige Änderung.

Kreuz- und Be-Vorzeichen werden an die Note angehängt, und zwar steht '#' für Kreuz und '-' für Be. Das wäre eigentlich alles, was ihr dazu braucht, um eure Lieder in QBasic umzusetzen. Hier noch eine Liste mit weiteren PLAY-Dingsbums (eigentlich sind es auch Befehle) wie '<' und '>'.

```
<..... eine Oktave tiefer
>..... eine Oktave höher

mf..... spiele Musik im Vordergrund
mb..... spiele Musik im Hintergrund (Einlesen der Noten ist zu langsam!)

ml..... legato
ms..... staccato
mn..... normal

#..... Kreuz
-..... Be

lx..... Notenlänge 1/x
tx..... Tempo x
ox..... Oktave x
```

Nochmals zu den Notennamen: **ALLE MÜSSEN NACH DER REGEL EINGEGEBEN WERDEN!**

Solche Spezialitäten wie das Es versteht QBasic nicht. Habt ihr eigentlich schon 'mal versucht, 'c-' zu spielen? Daß diese Note dem 'b' der vorigen Oktave entspricht, kapiert der Computer nicht! Für ihn liegen Halbtöne nur auf den schwarzen Tasten.

Uff, das war 'ne Menge! Und weil's so schön ist, binden wir in unser Proggy noch Musik ein!!!!

```
'DAS MEGARECHENPROGRAMM!!!!
'... aus The Real Adok's Way to QBASIC.
```

```
'+++ Variablenzuweisungen
blau = 1           'Farbe Blau
gelb = 14          'Farbe Gelb
weiss = 7          'Farbe Weiß
hell = 8           'Zusatz für Helligkeit
```

```

'+++ Hauptprogramm
COLOR , blau           'Blauer Hintergrund
CLS                   'Bildschirm löschen

COLOR hell + weiss    'Eingabe der ersten Zahl
LOCATE 2, 2           '
PRINT "Wie lautet die erste Zahl";
COLOR gelb            '
INPUT zahl1          '

COLOR hell + weiss    'Eingabe der zweiten Zahl
LOCATE 5, 2           '
PRINT "Wie lautet die zweite Zahl";
COLOR gelb            '
INPUT zahl2          '

CLS                   'Bildschirm löschen
COLOR hell + weiss    'Erste Zahl ausgeben
LOCATE 2, 2           '
PRINT "Zahl 1:";      '
COLOR gelb            '
PRINT zahl1;          '

COLOR hell + weiss    'Zweite Zahl ausgeben
LOCATE , 60           '
PRINT "Zahl 2:";      '
COLOR gelb            '
PRINT zahl2           '

COLOR hell + weiss    'Verzierungsstrichlein ausgeben
LOCATE 3, 1           '
PRINT "-----";      '
PRINT "-----";      '
PRINT "-----";      '
PRINT "-----"        '

COLOR hell + weiss    'Ergebnis der Addition ausgeben
LOCATE 5, 2           '
PRINT "Addition:";    '
COLOR gelb            '
LOCATE , 17           '
PRINT zahl1 + zahl2   '

COLOR hell + weiss    'Ergebnis der Subtraktion ausgeben
LOCATE 6, 2           '
PRINT "Subtraktion:";
COLOR gelb            '
LOCATE , 17           '
PRINT zahl1 - zahl2   '

COLOR hell + weiss    'Ergebnis der Multiplikation ausgeben
LOCATE 7, 2           '
PRINT "Multiplikation:";
COLOR gelb            '
LOCATE , 17           '
PRINT zahl1 * zahl2   '

COLOR hell + weiss    'Ergebnis der Division ausgeben
LOCATE 8, 2           '
PRINT "Division:";    '
COLOR gelb            '
LOCATE , 17           '
PRINT zahl1 / zahl2   '

PLAY "t200 l8 o5 ddd p64 l2 g" 'Musik

'+++ Ende des Programms.

```

Das Programm ist mit seinen 3 KB eigentlich zum Abtippen schon gewaltig groß.

## KAPITEL 4

### Debugging in Quick Basic und Power Basic

Wer noch nie einen kleinen Programmfehler produziert hat, der werfe mit dem ersten Stein! Na, das wird aber lange dauern, bis der erste Stein geworfen wird!

Beginnen wir also, indem wir selbst einen Fehler produzieren. Gebt dazu folgende Programmzeile ein:

```
LINE (100, 100) - (150, )
```

Den Befehl *LINE* werden wir übrigens im Kapitel über die Grafikprogrammierung noch näher kennenlernen. Es sei jetzt nur gesagt, daß man mit ihm im Grafikmodus Linien zeichnen kann.

Wenn wir jetzt zur Bestätigung *ENTER* betätigen, meldet QBasic einen Fehler (Erwartet: Ausdruck).

Wir haben nun zwei Möglichkeiten. Zum einen können wir uns eine wenig aufschlußreiche Fehlererklärung ansehen, die jedoch zu kaum etwas nütze ist, weil es x Möglichkeiten gibt, dieselbe Fehlermeldung zu produzieren.

Die andere Möglichkeit ist, in den Editor zurückzukehren, um dort den Fehler zu suchen und zu korrigieren. Wir wählen jetzt diese Möglichkeit.

Da der Fehler während der Eingabe auftauchte, kann man daraus schließen, daß er sich in der gerade eingegebenen Programmzeile befinden muß. Die Kunst des Fehlerbehebens besteht nun darin, sich die Programmzeile noch einmal genau durchzulesen und bei den Befehlen und den Parametern zu prüfen, ob man sich an alle Regeln hielt.

Ihr kennt den Befehl *LINE* noch nicht und könnt daher auch nicht den Fehler feststellen. Deshalb verrate ich euch, was da falsch ist: Eine Zahl fehlt. Also schreibt zwischen den letzten Beistrich und die Klammer die Zahl '200'.

So, das war die eine Möglichkeit, wo Fehler auftreten können: bei der Programmeingabe. Glücklicherweise überprüfen QBasic und Quick Basic immer die aktuelle Zeile auf Eingabefehler, wodurch sich diese schneller feststellen lassen. Power Basic stellt diese Eingabefehler jedoch erst dann fest, nachdem man den Befehl zum Ausführen des Programms erteilt.

Von den logischen Fehlern einmal abgesehen, die keine Programmiersprache feststellen kann und auf die wir später zu sprechen kommen, sind Eingabefehler die häufigsten Fehler. Doch es können auch Laufzeitfehler, sprich Fehler während der Ausführung eines Programms, entstehen. QBasic-Benutzer sind hier besonders im Nachteil, weil QBasic nur ein Interpreter ist und im Gegensatz zu den Compilern einige Fehler erst während der Ausführung des jeweiligen Befehls entdeckt werden können.

Eine häufige Art von Laufzeitfehlern entsteht, wenn man Grafikbefehle benutzt, ohne vorher in den Grafikmodus umgeschaltet zu haben (wozu der Grafikmodus dient, werden wir später erfahren). Dies ist übrigens auch bei unserem Einzeiler-Programm so.

Also starten wir es, um diesen Laufzeitfehler zu produzieren. Wir erkennen, daß Laufzeitfehler im Grunde genommen genauso wie Eingabefehler behandelt werden, nur mit dem Unterschied, daß sich der Fehler nicht in der aktuellen Zeile befinden muß. Hier benötigen wir also etwas Logik, um herauszufinden, was nun falsch sei. Bei unserem Beispiel habe ich es ja schon verraten: Wir vergaßen, vorher in den Grafikmodus umzuschalten. Dies können wir korrigieren, indem wir eine Zeile vor den *LINE*-Befehl einfügen und dort

```
SCREEN 12
```

schreiben. Jetzt sieht das Programm folgendermaßen aus:

```
SCREEN 12
LINE (100, 100) - (150, 200)
```

Wenn wir es nun starten, erscheint eine nette, kleine Gerade auf dem Bildschirm. Hurra! Wir haben es geschafft!

Die häufigste Fehlerart sind aber logische Fehler. Dies sind die aus vielen professionellen Anwendungen bekannten Bugs, wegen denen das Programm nicht immer tut, was der Benutzer will!

Da bei diesen Fehlern die Befehle meist richtig geschrieben und den Regeln gültig angewendet werden, kann QBasic sie nicht finden. Abhilfe schaffen nur sorgfältiges Planen vor der eigentlichen Codierung oder strukturiertes Programmieren.

QBasic versucht aber, den Programmierer so weit wie möglich bei der Suche nach logischen Fehlern in seinen Programmen zu unterstützen. Dazu gibt's die Möglichkeit, mit *F9* Haltepunkte einzufügen. Wenn das Programm nun mit *F5* gestartet wird, wird es nur bis zu der Zeile, in der der Haltepunkt gesetzt ist, ausgeführt.

Danach kehrt QBasic in den Editor zurück, wo man nun kleine Änderungen und Fehlerkorrekturen durchführen kann. Durch einen erneuten Druck auf *F5* wird die Programmausführung von diesem Haltepunkt an bis zum nächsten fortgesetzt, während mit *Shift* und *F5* gleichzeitig das Programm von Anfang an neu gestartet wird.

Diese Haltepunkte sind besonders nützlich in Kombination mit den erweiterten Debugging-Funktionen von Quick Basic und Power Basic. Dort kann man nämlich im Menü *Debug* (QB) bzw. *Debug - Watch* (PB) die jeweils aktuellen Inhalte von Variablen dauerhaft auf dem Bildschirm des Editors anzeigen.

In Quick Basic heißt die Menüoption, die ihr wählen müßt, *Variable anzeigen*, in Power Basic *Add Watch*. Danach müßt ihr den Namen der Variable angeben, deren aktueller Wert ständig angezeigt werden soll.

Schon erscheinen der Name der Variablen und der aktuelle Wert auf dem Bildschirm! Jetzt könnt ihr entweder das Programm starten oder mit dem Editieren fortfahren.

Mit den anderen Befehlen im angegebenen Menü lassen sich diese angezeigten Variablen wieder vom Bildschirm löschen. Die Variablen selbst werden dabei nicht aus dem Speicher gelöscht!!!

Nützlich ist auch die Funktion *Aktueller Wert anzeigen* von Quick Basic (in Power Basic müßte es dazu auch ein Äquivalent geben; wie es heißt, weiß ich aber momentan nicht), die den aktuellen Wert der Variablen anzeigt, auf der sich der Cursor befindet. Mit einem Mausklick läßt sich diese Variable dann in die Liste der zu anzeigenden Variablen aufnehmen.

## KAPITEL 5

### Blöcke

Nun möchte ich einen Befehl besprechen, der etwas aus der Reihe fällt. Außerdem ist er unser erster Block-Befehl, was heißen soll, daß er eigentlich aus zwei Befehlen besteht.

Bei dem Befehl handelt es sich konkret - tatatata - um *DO/LOOP*. Wir haben ihn schon früher einmal in einem Beispielprogramm kennengelernt. Er ist übrigens einer der vielseitigsten Befehle, die es gibt (diesmal auf *ALLE* Programmiersprachen bezogen).

Als "nackter" Befehl ohne Parameter, bedeutet er, daß der Programmteil zwischen den beiden Schlüsselwörtern *DO* und *LOOP* unendlich oft wiederholt wird. Man nennt dies eine Endlosschleife.

Solche Endlosschleifen sind nicht besonders ratsam, weil man sie nur durch gleichzeitiges Drücken von *STRG* und *PAUSE* abbrechen kann - wenn überhaupt. Aber keine Angst, bei QBasic funktioniert es bis auf einigen wenigen Ausnahmen immer!

Bei Power Basic ist das wieder etwas anderes, weil dieses keine Programme in der Vorgangsweise eines Interpreters übersetzen kann, sondern nur wie ein Compiler. Compiler sind meistens sowieso besser, aber das ist ein anderes Thema.

Unser kleines Experiment, das jetzt folgen wird, müßte auch in Power Basic kein Problem sein:

```
'Endlosschleife, kann nur mit STRG und PAUSE abgebrochen werden!

COLOR 14, 9
CLS
DO                                     'Anfang der Schleife.
  PRINT "Hallo, Leute!"
  PRINT "Hi, PC-Heimwerker!"
  PRINT "Viele Grüße von QBASIC!"
  Stopper = TIMER                       'Kleine Stoppschleife, weil das ganze
  DO                                     'sonst zu schnell wird.
  LOOP UNTIL TIMER > Stopper + .5
LOOP                                     'Ende der Schleife.
```

So, nehmen wir dieses zwölfzeilige Proggy genauer unter die Lupe.

*Schleife* ist übrigens eine Bezeichnung für eine spezielle Art von Blöcken, zu denen auch *DO/LOOP* gehört.

In der dritten Zeile wird die Farbkombination gewählt, in der vierten der Bildschirm gelöscht.

Die fünfte Zeile beinhaltet jetzt den neuen Befehl: *DO*. Er setzt den Beginn der Schleife.

Die Zeilen 6 bis 11 beinhalten den eigentlichen Code, der unendlich oft ausgeführt wird. Ich habe diese Zeilen zugunsten der Übersicht eingerückt. Man sollte sich angewöhnen, bei Blöcken immer den eigentlichen Code zwischen Blockanfang und Blockende einzurücken, weil es sonst sehr schnell unübersichtlich wird.

In der Schleife befindet sich außerdem eine weitere Schleife, die ein markantes Merkmal von Blöcken aufweist: Sie lassen sich ineinander verschachteln. Das innerste *LOOP* gehört zum innersten *DO*, das zweitinnerste *LOOP* zum zweitinnersten *DO* usw. bis zum äußersten *LOOP*, das zum äußersten *DO* gehört.

Theoretisch kann man Blöcke unendlich oft verschachteln! Hier haben wir allerdings nur zwei "Ebenen" verwendet: Das innere *LOOP* gehört zum inneren *DO*, das äußere *LOOP* zum äußeren *DO*.

Was *UNTIL* und *TIMER* bedeuten, braucht ihr noch nicht zu wissen (*UNTIL* wird schon bald erklärt), jedenfalls sei gesagt, daß die innere Schleife dazu dient, zwischen den einzelnen Textausgaben eine kleine Pause zu machen.

Das *LOOP* in Zeile 12 schließlich veranlaßt QBASIC, zurück in die Zeile mit *DO* zu springen. Da dort keine Abbruchsbedingungen zu finden sind, wird der Code innerhalb der Schleife wiederholt ausgeführt. Dann kommt QBASIC wieder zu *LOOP*, springt nach *DO*, führt den Code innerhalb der Schleife aus usw.

Was hat es nun mit diesem *UNTIL* auf sich?

Die Schlüsselwörter *UNTIL* und *WHILE* können nach *DO* oder *LOOP* stehen und geben eine Abbruchsbedingung an. Wenn diese erfüllt ist, wird bei *UNTIL* die *DO/LOOP*-Schleife abgebrochen und der nächste Befehl nach *LOOP* ausgeführt (*UNTIL*).

Bei *WHILE* ist das ganze umgekehrt: Wenn die Bedingung *NICHT* erfüllt ist, wird die Schleife abgebrochen, ansonsten wird die Ausführung fortgesetzt.

Da ein Beispiel mehr sagt als tausend Worte, hier eine kleine Anwendung für *UNTIL*. Sie zählt von 0 bis auf 10 hinauf und gibt dabei die Zahl und einen Text aus. Mittlerweile sind wir übrigens schon bei unserem zehnten Beispielprogramm angelangt. Jubiläum!!!!

```
'Zählschleife

COLOR 14, 1
CLS

Zaehler = 0
DO
  PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."
  Zaehler = Zaehler + 1
```

```
LOOP UNTIL Zaehler = 10
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."
```

Das Interessante beginnt ab Zeile 6. Dort wird der Schleifenzähler auf 0 gesetzt. Eigentlich unwichtig, weil alle Zahlenvariablen vor einer Zuweisung 0 enthalten, doch zur Übersichtlichkeit machen wir es hier.

Danach wird mit *DO* der Schleifenkopf eingeleitet. Hier ist nichts Neues dazugekommen.

In der achten Zeile wird nun ausgegeben, zum wievielten Mal die Schleife durchlaufen wird. Eigentlich ist das etwas unlogisch, weil beim ersten Schleifendurchlauf ausgegeben wird, daß die Schleife zum nullten Mal durchlaufen wird! Aber das könnt ihr ja selbst sehr einfach auf zwei Arten ändern. Wie, das möchte ich nicht verraten. Macht es doch als Hausaufgabe!

In der neunten Zeile wird der Schleifenzähler um 1 erhöht. Und jetzt kommt's!

In der zehnten Zeile wird noch zusätzlich neben *LOOP* das Schlüsselwort *UNTIL* verwendet. *Zaehler = 10* ist nun so ein Vergleich, wie wir beim *IF/THEN/ELSE/ENDIF*-Block viele kennenlernen werden. Dieser hier fragt ab, ob der Inhalt der Variablen *Zaehler* 10 ist. Wenn ja, dann wird die Schleife abgebrochen und die *PRINT*-Anweisung, die sich in der elften Zeile befindet, ausgeführt.

Ansonsten wird das ganze nach *DO* verzweigt, und die Schleife wird erneut ausgeführt.

Diesmal ist jedoch die Variable *Zaehler* schon um 1 größer als beim letzten Durchlauf. Nun wird ausgegeben, daß die Schleife das erste Mal durchlaufen werde (fälschlicherweise, wie wir wissen) und das ganze geht weiter, bis der Zähler auf 10 erhöht wird. Dann wird die Schleife abgebrochen, und wir kommen zur abschließenden Textausgabe.

Diese Art der Zählschleife ist schon etwas aufwendig. Noch aufwendiger ist die Zählschleife mit *WHILE*.

```
'Zählschleife, Version mit WHILE

COLOR 14, 1
CLS

Zaehler = 0
DO
  PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."
  Zaehler = Zaehler + 1
LOOP WHILE Zaehler < 10
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."
```

Es gibt in QBasic glücklicherweise eine Schleife, die extra für Zählaufgaben gedacht ist: die *FOR/NEXT*-Schleife. Hier gleich ein Beispielprogramm:

```
'Zählschleife mit FOR/NEXT

COLOR 14, 1
CLS

FOR Zaehler = 1 TO 10 STEP 1
  PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."
NEXT
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."
```

Wie ihr seht, erspart das Verwenden der *FOR/NEXT*-Schleife eine Menge Schreibarbeit.

Die 6. Zeile ist der Schleifenanfang (wie *DO*) und die 8. das Schleifenende (wie *LOOP*).

In der 6. Zeile wird angegeben, daß die Variable *Zaehler* als Schleifenzähler verwendet werden soll. Der Startwert soll 1 und der Endwert 10 sein. *STEP* gibt die Schrittweite an. Wenn *STEP* weggelassen wird, ist die Schrittweite automatisch 1.

Was bewirkt nun der Schleifenanfang? Beim ersten Durchlauf wird der Schleifenzähler auf den Startwert gesetzt. Danach werden alle Befehle zwischen *FOR* und *NEXT* ausgeführt. *NEXT* erhöht nun den

Schleifenzähler um die Schrittweite. Wenn der Schleifenzähler gleich oder größer dem Endwert ist, dann wird die Schleife abgebrochen und der nächste Befehl ausgeführt. Ansonsten wird zu *FOR* zurückgesprungen und die Befehle zwischen *FOR* und *NEXT* erneut ausgeführt.

Ihr seht, das ganze verhält sich so ähnlich wie *DO/LOOP*. Auch Verschachtelung ist möglich. Sogar verschiedenartige Schleifenarten lassen sich verschachteln!

Das gilt auch für den *IF/THEN/ELSE/ENDIF*-Block, obwohl dieser keine Schleife, sondern eine Verzweigung ist. Wir werden diesen etwas umfangreicheren, aber sehr wichtigen Befehl jetzt besprechen.

Der *IF/THEN/ELSE/ENDIF*-Block dient dazu, Verzweigungen zu realisieren. Wenn eine bestimmte Bedingung erfüllt ist, wird der Programmteil zwischen *THEN* und *ENDIF* ausgeführt. Ansonsten wird zu dem Befehl hinter *ENDIF* gesprungen.

Die Bedingung ist meist ein Vergleich zweier Variablen oder einer Variablen und eines Werts. Ein Beispielprogramm demonstriert die möglichen Varianten.

```
'Vergleiche
COLOR 14, 1
CLS

PRINT "Gib eine Zahl ein!"
INPUT Zahl

PRINT "Diese Zahl erfüllt folgende Bedingungen:"
IF Zahl < 100 THEN
  PRINT "Sie ist kleiner als 100."
END IF
IF Zahl <= 100 THEN
  PRINT "Sie ist kleiner oder gleich 100."
END IF
IF Zahl = 100 THEN
  PRINT "Sie ist gleich 100."
END IF
IF Zahl >= 100 THEN
  PRINT "Sie ist größer oder gleich 100."
END IF
IF Zahl > 100 THEN
  PRINT "Sie ist größer als 100."
END IF
```

Jetzt müssten die meisten Vergleichoperatoren klar sein. Größer-oder-Gleich und Kleiner-oder-Gleich werden zwar nur bei Fließkommaoperationen benötigt (und auch da nicht so oft), doch der Vollständigkeit halber listete ich sie auch auf.

Dann gibt es da noch so nette kleine Befehle wie *AND*, *OR* usw. Mit ihrer Hilfe lassen sich mehrere Bedingungen miteinander verketteten.

Wenn zwei Bedingungen mit *AND* verknüpft werden, wird der zwischen *THEN* und *ENDIF* stehende Code nur dann abgearbeitet, wenn beide Bedingungen erfüllt sind. Richtig, das lässt sich auch durch zwei verschachtelte *IF/THEN/ELSE/ENDIF*-Blöcke realisieren! Außerdem werden verschachtelte Blöcke etwas schneller abgearbeitet als *AND*-Verknüpfungen, weshalb von letzteren eher abzuraten ist.

*OR* macht schon mehr Sinn. Hier wird der Code hinter *THEN* schon dann abgearbeitet, wenn mindestens eine Bedingung erfüllt ist.

*XOR* bedeutet woviel wie 'Entweder-Oder'. Nomen est Omen! Der hinter *THEN* stehende Code wird hier nur ausgeführt, wenn nur eine Bedingung erfüllt ist. Wenn beide oder gar keine erfüllt sind, springt QBASIC zu dem Befehl nach *ENDIF*.

Um das ganze zu demonstrieren, wieder ein Beispielprogramm:

```
'Vergleiche mit AND, OR und XOR
```

```

COLOR 14, 1
CLS

PRINT "Gib die erste Zahl ein!"
INPUT Zahl1

PRINT "Gib die zweite Zahl ein!"
INPUT Zahl2

PRINT "Es werden folgende Bedingungen erfüllt:"
IF Zahl1 = 100 OR Zahl2 = 100 THEN
  PRINT "Mindestens eine der beiden Zahlen ist 100."
END IF
IF Zahl1 = 100 AND Zahl2 = 100 THEN
  PRINT "Beide Zahlen sind 100."
END IF
IF Zahl1 = 100 XOR Zahl2 = 100 THEN
  PRINT "Genau eine der beiden Zahlen ist 100."
END IF
IF Zahl1 <> 100 AND Zahl2 <> 100 THEN
  PRINT "Keine der beiden Zahlen ist 100."
END IF

```

Ach ja, 'Ungleich' wird in QBasic wie '<>' geschrieben. Das hatte ich vergessen, euch zu sagen.

Wozu ist nun der *ELSE*-Zweig gut? Normalerweise wird, wenn die Bedingung nicht erfüllt ist, die Programmausführung direkt beim nächsten Befehl hinter *ENDIF* fortgesetzt. Kommt nun ein *ELSE*-Zweig vor, wird bei einer nicht erfüllten Bedingung zuerst der Code zwischen *ELSE* und *ENDIF* ausgeführt! (Bei einer erfüllten Bedingung wird dann nur der Code zwischen *THEN* und *ELSE* ausgeführt.)

Das ganze läßt sich mit Worten ziemlich schwer erklären, doch in Wirklichkeit ist es überhaupt nicht kompliziert. Am besten, wir nehmen wieder ein Beispielprogramm zur Hand (schon KURS0015.BAS!).

```

'Vergleiche mit ELSE

COLOR 14, 1
CLS

PRINT "Gib eine Zahl ein!"
INPUT Zahl1

PRINT "Gib eine zweite Zahl ein!"
INPUT Zahl2

PRINT
IF Zahl1 = Zahl2 THEN
  PRINT "Die beiden Zahlen sind identisch."
ELSE
  PRINT "Die beiden Zahlen sind NICHT identisch."
END IF

```

Zum Spaß habe ich dieses Programm optimiert. Die optimierte Version ist zwar zum Verständnis dieses Kurses nicht notwendig, könnte aber sehr interessant sein.

```

'Vergleiche mit ELSE

COLOR 14, 1
CLS

PRINT "Gib eine Zahl ein!"
INPUT Zahl1

PRINT "Gib eine zweite Zahl ein!"
INPUT Zahl2

PRINT
PRINT "Die beiden Zahlen sind ";
IF Zahl1 <> Zahl2 THEN

```

```
PRINT "NICHT ";
END IF
PRINT "identisch."
```

Das wäre jetzt der *IF/THEN/ELSE/ENDIF*-Block.

Wie gesagt, läßt er sich wie alle Blöcke verschachteln. Diese Verschachtelung ist auch extrem wichtig für die strukturierte Programmierung von Adventures. Am Ende dieser Folge folgt ein Beispiel, jetzt aber noch schnell der *SELECT/CASE*-Block.

Wenn man schnell eine Variable auf mehrere Werte prüfen will, ist dieser Block wesentlich besser geeignet als ein *IF/THEN/ELSE/ENDIF*-Block. Wo sonst *'IF ... THEN'* steht, muß man *'SELECT CASE Variable'* schreiben. *'Variable'* steht hier für den Namen der Variablen, auf die sich die nachfolgenden Vergleiche beziehen.

Ein Vergleich wird immer mit *CASE* eingeleitet. Daneben steht der Wert, mit dem die am Blockanfang angegebene Variable verglichen werden soll. Es lassen sich auch mehrere, durch Beistriche getrennte Werte, neben *CASE* angeben, was dann soviel wie eine *OR*-Verknüpfung bedeutet.

Ist nun die Variable gleich einem der neben *CASE* angegebenen Werte, so wird der Code hinter *CASE* ausgeführt, bis zum nächsten *CASE*. Ansonsten wird gleich zum nächsten *CASE* gesprungen. Abgeschlossen wird der *SELECT/CASE*-Block mit *END SELECT*.

Am besten gleich ein Beispiel, wie das Ganze funktioniert. Es demonstriert gut, wie man die Eingabe bei Adventures mit dem *SELECT/CASE*-Block leicht realisieren kann.

```
'Einfaches Adventure-Spiel mit SELECT/CASE

COLOR 14, 1
CLS

PRINT "Willkommen zum Mysterious QBASIC Adventure. Du mußt immer die Zahl"
PRINT "eingeben, die neben der gewählten Option steht. 'aT' steht für"
PRINT "eine beliebige andere Taste."
PRINT
PRINT "Du stehst vor einem Schloß. Wagst Du Dich, es zu betreten?"
PRINT
PRINT " ( 1) Ja"
PRINT " (aT) Nö"
PRINT

LINE INPUT Eingabe$
CLS
SELECT CASE Eingabe$
CASE "1"
PRINT "Jetzt bist Du im Schloß. Du hast die Wahl, ob Du nach Westen oder"
PRINT "nach Osten gehst. Natürlich kannst Du auch das Schloß verlassen"
PRINT "(Feigling!) Wofür entscheidest Du Dich, oh edler Ritter?"
PRINT
PRINT " ( 1) Westen"
PRINT " ( 2) Osten"
PRINT " (aT) Schloß verlassen"
PRINT
LINE INPUT Eingabe$
CLS
SELECT CASE Eingabe$
CASE "1"
PRINT "Da ist ein böser Drache, den Du, edler Ritter, mit Deinem Schwert"
PRINT "zu Hackfleisch verarbeitest! Du hast die holde Prinzessin"
PRINT "gerettet und darfst sie jetzt zur Belohnung zur Frau nehmen (und"
PRINT "mit ihr 'ne Menge Spaß haben)! Ende."
END
CASE "2"
PRINT "Oh, Du armer Held. Kaum gehst Du nach Osten, fällst Du in einen"
PRINT "bodenlosen Abgrund! Und wenn er nicht gestorben ist, dann fällt"
PRINT "er noch heute. Ende."
END
CASE ELSE
PRINT "Kaum hast Du das Schloß verlassen, fällt das Tor hinter Dir zu."
```

```

    PRINT "Du hast Deine Chance vertan, denn 'rein kannst Du nimmer! Ende."
    END
END SELECT
CASE ELSE
    PRINT "Na gut, dann nicht. Starte lieber Excel und erledige langweilige"
    PRINT "Tabellenkalkulationen! Ende."
END SELECT

```

Hier wird auch die Verschachtelung demonstriert. *CASE ELSE* entspricht übrigens dem *ELSE* des *IF/THEN/ELSE/ENDIF*-Blocks, und *END* beendet das Programm an einer beliebigen Stelle.

Hier nun das ganze mit dem *IF/THEN/ELSE/ENDIF*-Block.

```

'Einfaches Adventure-Spiel mit IF/THEN/ELSE/ENDIF

COLOR 14, 1
CLS

PRINT "Willkommen zum Mysterious QBASIC Adventure. Du mußt immer die Zahl"
PRINT "eingeben, die neben der gewählten Option steht. 'aT' steht für"
PRINT "eine beliebige andere Taste."
PRINT
PRINT "Du stehst vor einem Schloß. Wagst Du Dich, es zu betreten?"
PRINT
PRINT " ( 1) Ja"
PRINT " (aT) Nö"
PRINT

LINE INPUT Eingabe$
CLS
IF Eingabe$ = "1" THEN
    PRINT "Jetzt bist Du im Schloß. Du hast die Wahl, ob Du nach Westen oder"
    PRINT "nach Osten gehst. Natürlich kannst Du auch das Schloß verlassen"
    PRINT "(Feigling!) Wofür entscheidest Du Dich, oh edler Ritter?"
    PRINT
    PRINT " ( 1) Westen"
    PRINT " ( 2) Osten"
    PRINT " (aT) Schloß verlassen"
    PRINT
    LINE INPUT Eingabe$
    CLS
    IF Eingabe$ = "1" THEN
        PRINT "Da ist ein böser Drache, den Du, edler Ritter, mit Deinem Schwert"
        PRINT "zu Hackfleisch verarbeitet! Du hast die holde Prinzessin"
        PRINT "gerettet und darfst sie jetzt zur Belohnung zur Frau nehmen (und"
        PRINT "mit ihr 'ne Menge Spaß haben)! Ende."
        END
    END IF
    IF Eingabe$ = "2" THEN
        PRINT "Oh, Du armer Held. Kaum gehst Du nach Osten, fällst Du in einen"
        PRINT "bodenlosen Abgrund! Und wenn er nicht gestorben ist, dann fällt er"
        PRINT "noch heute. Ende."
        END
    ELSE
        PRINT "Kaum hast Du das Schloß verlassen, fällt das Tor hinter Dir zu. Du"
        PRINT "hast Deine Chance vertan, denn 'rein kannst Du nimmer! Ende."
        END
    END IF
ELSE
    PRINT "Na gut, dann nicht. Starte lieber Excel und erledige langweilige"
    PRINT "Tabellenkalkulationen! Ende."
END IF

```

Wie man sieht, ist das Listing zwar etwas kürzer, aber komplizierter! Letztendlich bleibt aber dem Programmierer überlassen, ob er *SELECT/CASE* oder *IF/THEN/ELSE/ENDIF* verwendet. Ich persönlich bevorzuge *SELECT/CASE*.

## Lösungen zu Kapitel 5

**Aufgabe 1:** Es gibt zwei Möglichkeiten, um den Schleifenzähler am Anfang auf 1 und nicht auf 0 zu setzen. Die naheliegendere ist, bei der Zuweisung vor der Schleife ganz einfach 0 in 1 zu verändern. Das Listing sieht dann folgendermaßen aus:

```
'Zählschleife

COLOR 14, 1
CLS

Zaehler = 1
DO
  PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."
  Zaehler = Zaehler + 1
LOOP UNTIL Zaehler = 10
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."
```

Na, war doch ganz einfach! Es gibt aber noch eine zweite, nicht ganz so naheliegende Möglichkeit: Man kann den Befehl mit der Erhöhung des Schleifenzählers gleich hinter *DO* zu schreiben! Das sieht dann so aus:

```
'Zählschleife

COLOR 14, 1
CLS

Zaehler = 0
DO
  Zaehler = Zaehler + 1
  PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."
LOOP UNTIL Zaehler = 10
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."
```

Diese Variante hat allerdings eine Besonderheit: Wenn wir das Programm starten, erkennen wir, daß die Schleife nun auch ein zehntes Mal durchlaufen wird. Man sollte sich auch beim Coden in der Praxis klar sein, daß es unter Umständen zu anderen Ergebnissen führt, wenn man eine Wertzuweisung - und das ist eine Werterhöhung im Grunde genommen auch - an den Anfang der Schleife schreibt, als wenn sie am Ende steht! Es ist oft wichtig, diesen Umstand zu kennen.

## KAPITEL 6

### Programmierung von Abenteuerspielen

Wenn ihr alle brav mitgelernt habt, erfüllt ihr jetzt schon alle Voraussetzungen, um kreativ zu sein! Wir werden zusätzlich einige neue Techniken kennenlernen, die auch in anderen Programmieraufgaben nützlich sein können.

Wenn man ein Adventure erstellen will, sollte man sich zunächst vor der Codierung einen groben Überblick über die Story verschaffen. Normalerweise ist ein Adventure eine fortlaufende Geschichte, bei der man an einigen Stellen in die Handlung eingreifen kann. Bei professionellen Adventures sind aber die Eingriffsmöglichkeiten so groß, daß das ganze zu einem Spiel wird.

Programme mit weniger Eingriffsmöglichkeiten nennen sich interaktive Filme. Die meisten von ihnen sind so primitiv, daß man nur ab und zu eine Richtungstaste drücken und hoffen darf, daß sie die richtige war!

Unser erstes Spiel soll etwas mehr Eingriffsmöglichkeiten als interaktive Filme, aber weniger als professionelle Adventures, bieten.

Oft werden in Adventures Gegenstände benutzt. Diese können wir in Variablen speichern. Wenn die jeweilige Variable 1 ist, dann besitzt die Person den Gegenstand, wenn sie 0 ist, dann nicht. Man könnte auch noch andere Werte einsetzen, z.B. 2: Der Gegenstand ist bereits benutzt worden, befindet sich aber immer noch im Besitz des Spielers. Der Phantasie des Programmieres sind keine Grenzen gesetzt (höchstens die des Arbeitsspeichers)!

Wie bereits im Beispiel aus der vorigen Folge gezeigt wurde, werden die Auswahlmöglichkeiten mit *PRINT* angezeigt. Danach wird die Wahl eingegeben und mit *SELECT/CASE* ausgewertet.

Um die Eingabe wenigstens etwas komfortabler zu gestalten, ordnen wir jeder bis auf die letzte Auswahlmöglichkeit eine Zahl zu. Der Benutzer muß dann nur die Zahl, die der gewünschten Auswahlmöglichkeit zugeordnet ist, eingeben. Für die letzte Auswahlmöglichkeit kann dann jede beliebige andere Zahl oder Zeichenkette eingegeben werden.

Da *LINE INPUT* hier geradezu verschwenderisch als Eingabebefehl ist (immerhin muß nur ein Zeichen eingegeben werden), benutzen wir einen anderen, sehr praktischen Befehl.

Eigentlich ist er kein Befehl, sondern eine Funktion. Den Unterschied zwischen Befehlen und Funktionen werden wir in Folge 10 genauer definieren. Hier sei nur gesagt, daß Funktionen normalerweise wie Wertzuweisungen behandelt werden. Man kann sie aber auch mit einem Befehl kombinieren, z.B. wäre '*PRINT INKEY\$*' durchaus möglich. *INKEY\$* ist auch eine Funktion, wir werden sie aber nicht in diesem Kurs besprechen.

Die Funktion, die wir jetzt besprechen, heißt *INPUT\$*. Die Syntax lautet

```
Zeichenkettenvariable$ = INPUT$(AnzahlZeichen)
```

Da wir bei Adventures den Anwender immer nur ein Zeichen eingeben lassen, benötigen wir die Form

```
Eingabe$ = INPUT$(1)
```

Die Funktion *INPUT\$(1)* wartet nun, bis der Benutzer eine Zahl oder ein Zeichen eingibt und speichert dieses danach in der Variablen *Eingabe\$*. Praktischer Nebeneffekt: Im Gegensatz zu *INPUT* und *LINE INPUT* wird weder der Cursor noch die Eingabe auf dem Bildschirm angezeigt, außer, wenn es durch eine spezielle Funktion des *LOCATE*-Befehls eingestellt wurde (siehe Online-Hilfe).

Deshalb eignet sich *INPUT\$(1)* auch gut, wenn man den Benutzer auffordert, eine beliebige Taste zu drücken, um fortzusetzen (so, wie es QBasic am Ende jeder Programmausführung tut).

Vom Befehl *SLEEP*, der in der Online-Hilfe beschrieben ist, rate ich dezent ab, weil die Eingabe dann im Tastaturpuffer gespeichert wird und dann für die nächste Eingabe übernommen wird. Auch die *SLEEP*-Warteschleifen sind mies, weil sich erstens nur ganze Sekunden angeben lassen und sie mit einem beliebigen Tastendruck abgebrochen werden können. Also beachtet diesen Befehl, den sich sicherlich schon einige aus der Online-Hilfe "herausgepickt" haben, nicht!

Weiters müssen wir bei Adventures oft im Programm "springen". Wenn wir z.B. ein Haus betreten und es später wieder verlassen, muß zu der Stelle, wo wir vor dem Haus stehen, gesprungen werde.

Auf die Gefahr hin, daß ich mich jetzt unbeliebt mache (vor allem wegen meiner Aussagen über die strukturierte Programmierung, die zu dem, was jetzt kommt, widersprüchlich sind): Am einfachsten sind solche Sprünge mit dem berühmt-berüchtigten Befehl *GOTO* realisierbar.

Dazu muß zuerst ein sogenanntes Label definiert werden, zu dem *GOTO* dann springen kann. Ein Label besteht aus einer Zeichenkette, diesmal aber ausnahmsweise nicht in Anführungszeichen eingeschlossen, und einem Doppelpunkt. Es muß in einer eigenen Zeile stehen.

Nach *GOTO* muß als Parameter der Name des Labels angegeben werden, denn es lassen sich natürlich auch mehrere Labels benutzen. Allerdings müssen die dann auch verschiedene Namen haben! (Ich weiß aus eigener Erfahrung, daß vieles, was für mich heute selbstverständlich ist, für den Anfänger unnötig kompliziert sein kann. Deshalb erkläre ich lieber alles genauer als notwendig, bevor ich einen Stapel von Fragen mit der Post geliefert bekomme.)

So, jetzt kennt ihr die Theorie. Nun wollen wir ein kleines Beispiel liefern.

```
'Die Reise zum Mond
'Stark gekürzte Version für The Real Adok's Way to QBASIC
```

```

COLOR 14, 1
CLS

'Vorspann
PRINT "Die Reise zum Mond - Das Gerhard-betriebene Raumschiff"
PRINT "Copyright (C) 1995-1996 by Adok Soft"
PRINT
PRINT "Stark gekürzte Version für The Real Adok's Way to QBASIC"
a$ = INPUT$(1)
CLS
PRINT "Clausi, Stefan, Philipp, Franzi, Gerhard, Marius, Zahra und Nele ";
PRINT "wollen gemeinsam auf den Mond fliegen! ";
PRINT "Clausi, das Genie, baute kurzerhand ein"
PRINT "Raumschiff namens Astrein Shuttle und ernannte sich zum Kommandanten."
PRINT "Das besondere am Astrein Shuttle ist sicherlich nicht seine Geschwin-"
PRINT "digkeit, die mit einigen Megametern pro Sekunde nicht sehr schnell ";
PRINT "ist,"
PRINT "sondern der Antrieb. ";
PRINT "Er funktioniert so: Gerhard hängt sich an das Raumschiff"
PRINT "an und lässt sich von Zahra füttern. ";
PRINT "Je mehr er (fr)ißt, desto mehr - pardon -"
PRINT "Abgase lässt er. Und diese Abgase treiben das Raumschiff an!"
a$ = INPUT$(1)
CLS
PRINT "Nochmals für alle: Die Besatzung besteht aus"
PRINT
PRINT "Clausi..... Kommandant"
PRINT "Franzi..... 1. Offizier"
PRINT "Philipp..... 2. Offizier"
PRINT "Stefan..... 3. Offizier"
PRINT "Marius..... Oberbiologe und Klowart"
PRINT "Nele..... Navigatorin"
PRINT "Gerhard..... Treibstoffherzeuger"
PRINT "Zahra..... Krankenschwester"
a$ = INPUT$(1)
CLS
PRINT "Du übernimmst die Rolle von Clausi, dem Kommandanten. ";
PRINT "Du und deine Besatzung"
PRINT "sind schon sehr lange (seit einer Minute!) im Weltraum mit dem ";
PRINT "Schiff unter - "
PRINT "wegs. ";
PRINT "Deine Aufgabe ist es, dich und deine Besatzung sicher zum Mond zu ";
PRINT "bringen. "
PRINT "Einfach? ";
PRINT "Ja, so klingt es. ";
PRINT "Ist es aber nicht..."
a$ = INPUT$(1)

'Das Spiel beginnt!
HauptHalleStart:
CLS
PRINT "Du bist nun in der Haupthalle des Astrein-Shuttles. Von hier aus ";
PRINT "kannst du"
PRINT "alle Räume des Raumschiffs erreichen und das Raumschiff sogar ";
PRINT "verlassen."
PRINT "Bevor du letzteres tust, rate ich dir, die notwendige"
PRINT "Ausrüstung zu holen..."
GOSUB Eingabe
CLS
SELECT CASE eingab$
CASE "1"
PRINT "Es ist niemand da, mit dem du sprechen könntest!"
CASE "2"
HheRetry1:
PRINT "Was willst du betrachten?"
PRINT " (1) Die Türen"
PRINT " (2) Den Raum"
PRINT " (3) Die Luft"
PRINT " (4) Das Schild"
e1$ = INPUT$(1)
CLS

```

```

SELECT CASE e1$
CASE "1"
  PRINT "Es sind fünf große Türen, die durch ein Laserfeld geschlossen ";
  PRINT "sind. Berührt"
  PRINT "man das Laserfeld, so wird es automatisch deaktiviert."
CASE "2"
  PRINT "Ein großer, leerer Raum, der Haupthalle genannt wird. An den ";
  PRINT "Wänden kleben"
  PRINT "alte Butterbrote, und die fünf Türen werden durch ein Laserfeld";
  PRINT "geschlossen. "
  PRINT "Wenn man diesen Raum mit einem bestimmten Klassenraum des ";
  PRINT "Goethe-Gymnasiums vergleicht, so ist eine gewisse Ähnlichkeit ";
  PRINT "erkennbar. "
CASE "3"
  PRINT "Das Sehorgan behauptet, sie wäre ein transparentes Nichts. Das ";
  PRINT "Geruchs- und"
  PRINT "das Atmungsorgan wissen es allerdings besser."
CASE "4"
  PRINT "Auf ihm steht 'EXIT'. Offensichtlich führt es zum Ausgang ";
  PRINT "dieses Raumschiffs."
  PRINT "Das solltest du, der Erbauer dieses Raumschiffs, eigentlich am ";
  PRINT "besten wissen!"
CASE ELSE
  GOTO HheRetry1
END SELECT
CASE "3"
HheRetry2:
  PRINT "Was willst du nehmen?"
  PRINT " (1) Laserfeld"
  PRINT " (2) Butterbrot"
  e1$ = INPUT$(1)
  CLS
  SELECT CASE e1$
  CASE "1"
    PRINT "Bist du lebensmüde oder was!? Wenn du so weitermachst, wirst du";
    PRINT " das Spiel nie"
    PRINT "schaffen!"
  CASE "2"
    PRINT "Igitt! Dieses grausame, an der Wand klebende, mit Butter ";
    PRINT "verzierte Brot ist"
    PRINT "ekelerregend! Laß es lieber kleben!"
  CASE ELSE
    GOTO HheRetry2
  END SELECT
CASE "4"
HheRetry3:
  PRINT "Wohin willst du gehen?"
  PRINT " (1) Tür 1"
  PRINT " (2) Tür 2"
  PRINT " (3) Tür 3"
  PRINT " (4) Tür 4"
  PRINT " (5) Tür 5"
  e1$ = INPUT$(1)
  CLS
  SELECT CASE e1$
  CASE "1", "2", "4"
    PRINT "Diesen Raum kannst du nur in der ungekürzten Fassung betreten!"
  CASE "3"
    GOTO KommandoraumStart
  CASE "5"
    GOTO OffizierszimmerStart
  CASE ELSE
    GOTO HheRetry3
  END SELECT
CASE ELSE
  GOTO HaupthalleStart
END SELECT
a$ = INPUT$(1)
GOTO HaupthalleStart

KommandoraumStart:
CLS

```

```

PRINT "Dies ist der Kommandoraum des Astrein Shuttles. Vor dir ist ein ";
PRINT "riesiges "
PRINT "Steuerpult, neben dem Nele sitzt, die vergeblich versucht, es zu ";
PRINT "bedienen. "
PRINT "Dahinter befindet sich in diesem Raum ein riesiges Fenster."
GOSUB Eingabe
CLS
SELECT CASE eingab$
CASE "1"
KreRetry1:
PRINT "Mit wem willst du sprechen?"
PRINT " (1) Nele"
PRINT " (2) Autopilot"
e1$ = INPUT$(1)
CLS
SELECT CASE e1$
CASE "1"
SELECT CASE MitNeleGeredet
CASE 1
PRINT "Nele: 'Konntest du den Autopiloten aktivieren?'"
PRINT "Clausi: 'Ich bin gerade dabei!'"
CASE 0
PRINT "Clausi: 'Gibt es Probleme?'"
PRINT "Nele: 'Ja, gewaltige! Wir sind vom Kurs abgekommen, und ";
PRINT "ich kann den"
PRINT " Autopiloten nicht mehr aktivieren!'"
PRINT "Clausi: 'Dann siehe doch in der Bedienungsanleitung nach!'"
PRINT "Nele: 'Ich habe sie verloren.'"
PRINT "Clausi: 'Wunderbar! Letzten Endes muß ich doch immer alles ";
PRINT "machen. Laß es"
PRINT " mich probieren!'"
MitNeleGeredet = 1
END SELECT
CASE "2"
PRINT "Clausi: 'Bitte, Herr Autopilot, fliegen Sie uns zum Mond!'"
PRINT
PRINT "....."
PRINT
PRINT "Keine Antwort. Tja, so leicht geht es nicht!"
CASE ELSE
GOTO KreRetry1
END SELECT
CASE "2"
KreRetry2:
PRINT "Was willst du betrachten?"
PRINT " (1) Nele"
PRINT " (2) Steuerpult"
PRINT " (3) Fenster"
PRINT " (4) Teppich"
e1$ = INPUT$(1)
CLS
SELECT CASE e1$
CASE "1"
PRINT "Nele, die Navigatorin des Astrein-Shuttles, sitzt gemütlich auf";
PRINT " ihrem Sessel, "
PRINT "hört sich mit ihrem Discman eine neue CD an und versucht ";
PRINT "herauszufinden, wie"
PRINT "man den Autopiloten des Astrein-Shuttles aktiviert."
CASE "2"
IF MitNeleGeredet = 1 AND Joystick = 5 THEN
PRINT "Du gibst den Code, den dir Franzi gesagt hat, ein."
PRINT "Plötzlich ruft Nele: 'Hurra! Du hast es geschafft! Du hast den";
PRINT " Autopiloten "
PRINT "aktiviert!'"
a$ = INPUT$(1)
CLS
PRINT "Das war die gekürzte Version von 'Die Reise zum Mond' für"
PRINT "The Real Adok's Way to QBASIC. See you in the Full Version!"
END
ELSE
PRINT "Das Steuerpult des Astrein-Shuttles besteht aus unzähligen ";
PRINT "Knöpfen, Hebeln,"

```

```

PRINT "Schaltern, Joysticks und Joypads. Sogar ein Computer mit ";
PRINT "vielen Spielen"
PRINT "ist eingebaut! Wozu, glaubt ihr, sollte man die Joysticks und ";
PRINT "Joypads sonst"
PRINT "brauchen, außer zur Unterhaltung der Besatzungsmitglieder?"
END IF
CASE "3"
PRINT "Ein riesiges Fenster, das den Blick zu dem, was sich außerhalb ";
PRINT "des Shuttles"
PRINT "befindet, gewährt! He, Moment mal! Da draußen sind auch Zahra ";
PRINT "und Gerhard!"
CASE "4"
PRINT "DEN TEPPICH ANSCHAUEN? Wie geht das, wenn hier gar kein ";
PRINT "Teppich ist?"
CASE ELSE
GOTO KreRetry2
END SELECT
CASE "3"
KreRetry3:
PRINT "Was willst du nehmen?"
PRINT " (1) Teppich"
PRINT " (2) Steuerpult"
PRINT " (3) Joystick"
e1$ = INPUT$(1)
CLS
SELECT CASE e1$
CASE "1"
PRINT "DEN TEPPICH NEHMEN? Wie geht das, wenn hier gar kein Teppich ";
PRINT "ist?"
CASE "2"
PRINT "Ein bißchen zu schwer, was? Ja, ein bißchen viel zu schwer!"
CASE "3"
IF Joystick < 2 THEN
PRINT "Du reit einen der vielen Joysticks aus dem Steuerpult heraus ";
PRINT "und steckst ihn"
PRINT "in die Tasche. Vielleicht kannst du ihn einmal brauchen?"
Joystick = 2
ELSE
PRINT "Du hast schon einen!"
END IF
CASE ELSE
GOTO KreRetry3
END SELECT
CASE "4"
KreRetry4:
PRINT "Wohin willst du gehen?"
PRINT " (1) Tr"
e1$ = INPUT$(1)
CLS
IF e1$ <> "1" THEN
GOTO KreRetry4
END IF
GOTO HaupthalleStart
CASE ELSE
GOTO KommandoraumStart
END SELECT
a$ = INPUT$(1)
GOTO KommandoraumStart

OffizierszimmerStart:
CLS
PRINT "Du befindest dich im Offizierskasino. Hier befinden sich die drei ";
PRINT "Offiziere "
PRINT "Franzi, Philipp und Stefan. Sie spielen gerade ein Schachspiel auf ";
PRINT "ihrem "
PRINT "Computer. (Knnen sie berhaupt Schach spielen!?) "
GOSUB Eingabe
CLS
SELECT CASE eingab$
CASE "1"
OzeRetry1:
PRINT "Mit wem willst du sprechen?"

```

```

PRINT " (1) Stefan"
PRINT " (2) Franzi"
PRINT " (3) Philipp"
e1$ = INPUT$(1)
CLS
SELECT CASE e1$
CASE "1"
  PRINT "Stefan: 'Hmmm, soll ich meinen König opfern, um die Dame zu ";
  PRINT "retten?'"
CASE "2"
  IF Joystick = 0 THEN
    PRINT "Franzi: 'Stefan, Philipp, seid nicht so gemein! Laßt mich ";
    PRINT "auch mitspielen!'"
    PRINT "Philipp: 'Dann bräuchten wir aber noch einen dritten Joystick,'"
    PRINT " und den haben"
    PRINT "          wir nicht!'"
  END IF
  IF Joystick = 5 THEN
    PRINT "Franzi: 'Bitte, gib mir die Tüte!'"
    PRINT "Clausi: 'Die Tüte gebe ich dir nicht!'"
    PRINT "Franzi: 'Wie ich schon sagte: Der Code heißt oben, rechts, ";
    PRINT "unten, links, oben"
    PRINT "          oder umgekehrt!'"
  END IF
  IF Joystick > 2 AND Joystick < 5 AND MitNeleGeredet <> 1 THEN
    PRINT "Franzi: 'Vielen Dank für den Joystick, Clausi!'"
    PRINT "Clausi: 'Gern geschehen!'"
  END IF
  IF Joystick = 4 AND MitNeleGeredet = 1 THEN
    PRINT "Clausi: 'Hey, Fratzi, wie wäre es mit dieser original ";
    PRINT "philippinischen "
    PRINT "          Plastiktüte? Sie ist von Philipps Bett ";
    PRINT "hinuntergefallen!'"
    PRINT "Franzi: 'Wirklich? Ein Gegenstand ist immer für ein Gerücht ";
    PRINT "gut! Gib her!'"
    PRINT "Clausi: 'Zuerst sagst du mir den Code!'"
    PRINT "Franzi: 'Na gut. Oben, rechts, unten, links, oben ... oder ";
    PRINT "umgekehrt?'"
    PRINT "Clausi: 'Du bist ja eine schöne Hilfe! Dafür bekommst du die ";
    PRINT "Tüte nicht!'"
    Joystick = 5
  END IF
  IF Joystick > 2 AND Joystick < 5 AND MitNeleGeredet = 1 THEN
    PRINT "Clausi: 'Hey, Fratzi, weißt du, wie man den Autopiloten ";
    PRINT "aktiviert?'"
    PRINT "Franzi: 'Nein! Aber ich habe die Bedienungsanleitung!'"
    PRINT "Clausi: 'Gib sie her!'"
    PRINT "Franzi: 'Hm... eigentlich sollte ich sie dir geben, denn du ";
    PRINT "bist erstens mein"
    PRINT "          Kommandant und zweitens hattest du mir den Joystick ";
    PRINT "vorhin gegeben."
    PRINT "          Trotzdem: Ich gebe dir die Anleitung nur, wenn du mir";
    PRINT "          noch ein"
    PRINT "          Geschenk machst! Mache mich zum Kommandanten!'"
    PRINT "Clausi: 'Das mache ich nicht!'"
    PRINT "Franzi: 'Dann mache etwas anderes!'"
    Joystick = 4
  END IF
  IF Joystick = 2 THEN
    PRINT "Franzi: 'Komm, Philipp, laß mich endlich mitspielen!'"
    PRINT "Philipp: 'Wie oft soll ich es dir noch sagen: Dazu brauchen ";
    PRINT "wir einen dritten"
    PRINT "          Joystick!'"
    PRINT "Clausi: 'Wie diesen hier, den ich in meiner Hand halte?'"
    PRINT "Philipp: 'Ja, genau einen wie den! Gib ihn her!'"
    PRINT "Clausi: 'Hier.'"
    PRINT "Franzi: 'Danke, Clausi! Du bist mein bester Freund!'"
    Joystick = 3
  END IF
CASE "3"
  PRINT "Philipp: 'Schach ist ein wahrhaft lustiges Spiel! Wenn ich es ";
  PRINT "bloß spielen"

```

```

    PRINT "           könnte...'"
CASE ELSE
    GOTO OzeRetry1
END SELECT
CASE "2"
OzeRetry2:
PRINT "Was willst du betrachten?"
PRINT " (1) Schachcomputer"
PRINT " (2) Franzi"
PRINT " (3) Philipp"
PRINT " (4) Stefan"
e1$ = INPUT$(1)
CLS
SELECT CASE e1$
CASE "1"
    PRINT "Ein riesengroßer Computer, mit dem man nur Schach spielen ";
    PRINT "kann, dafür aber"
    PRINT "zu dritt!"
CASE "2"
    PRINT "Dieser Junge nennt sich Franzi und ist erster Offizier. Seine ";
    PRINT "Spitznamen sind"
    PRINT "Fratzi, Intrigator und FBI. Letzteres bedeutet 'Flotter ";
    PRINT "blonder Insulaner'."
    PRINT "Naja, Marius, der diesen Spitznamen erfunden hatte, meinte mit ";
    PRINT "dem 'I' etwas"
    PRINT "anderes, aber wir wollen ja nicht frech sein."
CASE "3"
    PRINT "Dieser Junge nennt sich Philipp und ist zweiter Offizier. Er ";
    PRINT "haut auf die"
    PRINT "Drums wie die anderen auf ihre lieben Mitschüler."
CASE "4"
    PRINT "Dieser Junge nennt sich Stefan und ist dritter Offizier. Wie ";
    PRINT "man es auch an"
    PRINT "einer zweiten Stelle in diesem Spiel erfahren kann, nennt sich ";
    PRINT "dieses Genie"
    PRINT "auch Stex Mel. Eine seiner berühmtesten, vielzitierten ";
    PRINT "Aussagen:      'Philosophie'"
    PRINT "ist nichts anderes als eine Mischung aus Poesie und Dichtung.'"
CASE ELSE
    GOTO OzeRetry2
END SELECT
CASE "3"
OzeRetry3:
PRINT "Was willst du nehmen?"
PRINT " (1) Schachcomputer"
PRINT " (2) Philipp's Joystick"
PRINT " (3) Franzi"
e1$ = INPUT$(1)
CLS
SELECT CASE e1$
CASE "1"
    PRINT "Du hast doch zu Hause selber einen 'rumstehen!"
CASE "2"
    PRINT "Den gibt er nicht her!"
CASE "3"
    PRINT "Clausi: 'Fratzi, laß mich dich in meine Arme nehmen!'"
    PRINT "Franzi: 'Du willst mich wohl AUF den Arm nehmen!?''"
CASE ELSE
    GOTO OzeRetry3
END SELECT
CASE "4"
    PRINT "Wohin willst du gehen?"
    PRINT " (1) Tür"
OzeRetry4:
e1$ = INPUT$(1)
IF e1$ <> "1" THEN
    GOTO OzeRetry4
END IF
GOTO HaupthalleStart
CASE ELSE
    GOTO OffizierszimmerStart
END SELECT

```

```

a$ = INPUT$(1)
GOTO OffizierszimmerStart

Eingabe:
PRINT " (1) Sprich"
PRINT " (2) Betrachte"
PRINT " (3) Nimm"
PRINT " (4) Gehe zu"
PRINT " (5) Ende"
eingab$ = INPUT$(1)
IF eingab$ = "5" THEN
  END
END IF
RETURN

```

"Das soll ein kleines Beispiel sein?" wird sich so mancher jetzt fragen. Ja, mit seinen 443 Zeilen, die gut 14 KB Speicherplatz belegen, ist dieses Beispielprogramm schon sehr groß. Aber für ein voll funktionsfähiges Adventure mit allem Drum und Dran bis auf Grafik, Maussteuerung und Sound ist es geradezu winzig!

Zu dem Beispielprogramm noch einige Erläuterungen:

- *Joystick*, *MitNeleGeredet* etc. sind Variablen, in denen gespeichert wird, wie weit man im Spiel fortgeschritten ist, was man schon alles gemacht hat usw. Wenn ich schon früher gewußt hätte, daß 'Die Reise zum Mond' in einem Kurs veröffentlicht wird, hätte ich es eindeutiger und übersichtlicher gemacht.
- Der Befehl *GOSUB* entspricht von der Verwendung her im großen und ganzen *GOTO*, nur wird, wenn QBasic auf *RETURN* stößt, zu der Zeile, in der *GOSUB* steht, zurückgesprungen.

## KAPITEL 7

### Grafikprogrammierung

Uff, das wird umfangreich sein! Fangen wir also am besten gleich an.

Wenn wir die Grafikfunktionen nützen wollen, müssen wir zunächst mit dem *SCREEN*-Befehl in den Grafikmodus umschalten. Da es verschiedene Grafikmodi gibt, muß als Parameter die Nummer des gewünschten Modi angegeben werden. Die gebräuchlichsten Grafikmodi sind:

```

0.....Textmodus, 16 Farben, mehrere Bildschirmseiten, keine Grafik.
9.....EGA-Grafik, Grafikauflösung 640x350 Pixel, 16 Farben, mehrere
  Bildschirmseiten, nicht so schöne Farben wie in Modus 12.
12.....VGA-Grafik, Grafikauflösung 640x480 Pixel, 16 Farben, eine
  Bildschirmseite.
13.....VGA-Grafik, Grafikauflösung 320x200 Pixel, 256 Farben, eine
  Bildschirmseite, in Power Basic nicht anwählbar!

```

Dazu bedarf es einiger Klärung.

Obwohl der Text auch in den Grafikmodi angezeigt werden kann, wird trotzdem oft auch weiterhin der reine Textmodus (0) verwendet. Das liegt daran, daß die Textauflösung im reinen Textmodus mit 80x25 Zeichen sehr hoch ist, wobei sie in manchen Grafikmodi wesentlich kleiner ist.

Beispielsweise lassen sich im Modus 13 nur 40 Zeichen pro Zeile darstellen. Nur im Grafikmodus 12 ist sie mit 80x30 Zeichen sogar etwas besser als im reinen Textmodus. Man könnte sich jetzt fragen, warum nicht gleich der Modus 12 genommen wird. Dafür gibt es zwei Gründe. Zum einen wird eine VGA-Grafikkarte benötigt (und die hat nicht jeder), und zum anderen ist die Textausgabe im Modus 12 sehr langsam. Aber jetzt Schluß damit, obwohl ich diese kleinen Exkurse ab und zu einbauen muß, damit die Materie für jeden verständlich wird. Fahren wir mit dem Kurs fort.

Die Grafikauflösung gibt an, aus wievielen Punkten der Bildschirm aufgebaut ist. Ein Wert von 640x350 gibt beispielsweise an, daß sich in der horizontalen Richtung pro Zeile 640 Bildpunkte ansprechen lassen, und es 350 Zeilen gibt.

Der Wert bei den Farben gibt die Anzahl der verschiedenen Farben an, die sich gleichzeitig darstellen lassen.

Was 'Bildschirmseiten' bedeutet, ist für uns nicht von Bedeutung. Ich habe sie nur der Vollständigkeit halber angeführt.

So, genug mit dem Gelaber. Ich veröffentliche nun ein Beispielprogramm, damit das ganze verständlicher wird. Es wird der Grafikbefehl *PSET* verwendet. Mit seiner Hilfe lassen sich Bildpunkte in einer beliebigen Farbe "setzen". Die Farbe kann entweder direkt im Parameter gewählt werden, oder sie muß vorher durch *COLOR* festgelegt werden. Die genaue Syntax lautet:

```
PSET (XKoordinateDesBildpunkts, YKoordinateDesBildpunkts), Farbe
```

Damit keine Probleme auftauchen: Bei allen Grafikbefehlen muß immer zuerst die X-Koordinate angegeben werden. Bei *LOCATE* war das ja genau umgekehrt, wie wir es (hoffentlich) noch bestens wissen.

```
'PSET-Demo
SCREEN 12                'VGA-Grafikmodus 640x480 gewählt
COLOR 13                'Pink
PSET (100, 100)         'Farbe vorher gewählt
PSET (320, 200), 14    'Farbe direkt angegeben
```

Wir werden in unseren Beispielprogrammen immer im Grafikmodus 12 arbeiten. Falls ihr nur eine EGA-Grafikkarte besitzt, benutzt einfach den ähnlichen Grafikmodus 9.

Es gibt dann noch den Befehl *PRESET*. Er funktioniert genauso wie *PSET*, nur wird, wenn die Farbe nicht direkt angegeben wird, die aktuelle Hintergrundfarbe verwendet.

Der nächste wichtige Grafikbefehl ist *LINE*. Mit seiner Hilfe lassen sich, wie der Name sagt, Linien zeichnen. Das ist übrigens nur eine seiner vielen Anwendungsmöglichkeiten, wie wir gleich sehen werden.

Die Syntax entspricht der von *PSET*, nur müssen zwei Koordinaten angegeben werden. Gleich ein Beispielprogramm:

```
'LINE-Demo
SCREEN 12                'VGA-Grafikmodus 640x480 gewählt
COLOR 13                'Pink
LINE (100, 100)-(320, 200) 'Farbe vorher gewählt
LINE (80, 400)-(420, 400), 14 'Farbe direkt angegeben
```

Experimentiert ein wenig mit den Koordinaten, um die Anwendung dieses Befehls zu erlernen.

Kommen wir nun zur anderen Anwendung von *LINE*. Mit *LINE* lassen sich auch Rechtecke zeichnen. Man kann sogar angeben, ob die Rechtecke ausgefüllt werden sollen oder nicht. Falls sie ausgefüllt werden sollen, muß man nach Angabe der Farbe ', BF' schreiben. Falls nicht, genügt ', B'.

Wenn ihr die Farbe nicht direkt angeben, sondern die in *COLOR* angegebene übernehmen wollt, laßt sie einfach weg! Der Beistrich muß aber schon geschrieben werden.

Dazu gleich ein Beispiel.

```
'LINE-Demo mit B- und BF-Option
SCREEN 12                'VGA-Grafikmodus 640x480 gewählt
COLOR 13                'Pink
```

```

LINE (10, 200)-(600, 400), , B      'Nicht ausgefülltes Rechteck

COLOR 15                             'Leuchtend-Weiß
LOCATE 24, 1                          'Meldung ausgeben
PRINT "Nicht ausgefülltes Rechteck"
PRINT "Beliebige Taste drücken..."
a$ = INPUT$(1)

CLS                                   'Bildschirm löschen
COLOR 13                               'Pink
LINE (10, 200)-(600, 400), , BF      'Ausgefülltes Rechteck

COLOR 15                             'Leuchtend-Weiß
LOCATE 24, 1                          'Meldung ausgeben
PRINT "Ausgefülltes Rechteck"
PRINT "Beliebige Taste drücken..."
a$ = INPUT$(1)

```

Wie ihr seht, wird in unserem Beispielprogramm zwischen den Grafik- und den Textausgaben immer die Farbe umgeschaltet. Da würde eine direkte Farbanwahl doch Sinn machen! Ich habe das deshalb gleich realisiert. Das Ergebnis lautet:

```

'LINE-Demo mit B- und BF-Option

SCREEN 12                             'VGA-Grafikmodus 640x480 gewählt

COLOR 15                             'Leuchtend-Weiß

LINE (10, 200)-(600, 400), 13, B     'Nicht ausgefülltes Rechteck

LOCATE 24, 1                          'Meldung ausgeben
PRINT "Nicht ausgefülltes Rechteck"
PRINT "Beliebige Taste drücken..."
a$ = INPUT$(1)

CLS                                   'Bildschirm löschen
LINE (10, 200)-(600, 400), 13, BF    'Ausgefülltes Rechteck

LOCATE 24, 1                          'Meldung ausgeben
PRINT "Ausgefülltes Rechteck"
PRINT "Beliebige Taste drücken..."
a$ = INPUT$(1)

```

Nun müßte euch klar sein, wozu eine direkte Farbanwahl gut ist: Wenn in einem Programm der Text in einer anderen Farbe als die Grafik ausgegeben werden soll, so ist es praktisch, die Textfarbe mit *COLOR* festzulegen und die Grafikfarbe immer direkt anzugeben.

Der nächste Befehl ist *CIRCLE*. Mit ihm lassen sich Kreise zeichnen. Als Parameter müssen der Mittelpunkt, der Radius und bei direkter Farbanwahl die Farbe angegeben werden.

Zu dem Resultat sei nur gesagt, daß QBASIC automatisch die Werte "nachkorrigiert", so daß der Kreis auch wie ein Kreis aussieht. Die Koordinaten stimmen deshalb nicht immer mit den Werten, die ihr durch den Mittelpunkt und den Radius errechnen könnt, überein. Wie immer, hier ein Beispiel.

```

'CIRCLE-Demo

'Variablen
MPunktX = 320
MPunktY = 200
Radius = 100

'Hauptprogramm
SCREEN 9                             'EGA-Grafikmodus 640x350

CIRCLE (MPunktX, MPunktY), Radius, 15 'Kreis zeichnen

'Beweis, daß die von Hand errechneten Koordinaten nicht mit den tatsächlichen
'übereinstimmen:
LINE (MPunktX - Radius, MPunktY)-(MPunktX + Radius, MPunktY), 14

```

```
LINE (MPunktX, MPunktY - Radius)-(MPunktX, MPunktY + Radius), 14
```

Bei dem Befehl *CIRCLE* gibt es keinen Fill-Parameter. Trotzdem lassen sich Kreise und andere Gebilde ausfüllen, und zwar mit dem Befehl *PAINT*.

Das ganze funktioniert so ähnlich wie die Fülloptionen in diversen Grafikprogrammen. Es müssen ein Startpunkt, bei dem das Füllen begonnen werden soll, die Ausfüllfarbe und die Randfarbe angegeben werden. Die Randfarbe gibt an, wo das Ausfüllen gestoppt werden soll. Man muß also eine Art "Käfig" in dieser Randfarbe zeichnen, bevor man *PAINT* verwendet. Sonst wird der ganze Bildschirm ausgefüllt.

Beim Ausfüllen eines Kreises sind diese Parameter sonnenklar: Startpunkt entspricht dem Mittelpunkt, Ausfüllfarbe und Randfarbe der Farbe des Kreises. Dazu gleich ein Beispiel.

```
' PAINT-Demo

SCREEN 12

LOCATE 2, 32
PRINT "Das ist PFLAUMI!"

CIRCLE (320, 200), 100, 9      'Kopf
PAINT (320, 200), 9, 9

CIRCLE (280, 166), 30, 14    'Linkes Auge
PAINT (280, 166), 14, 14

CIRCLE (360, 166), 30, 14    'Rechtes Auge
PAINT (360, 166), 14, 14

CIRCLE (280, 166), 10, 2     'Linke Pupille
PAINT (280, 166), 2, 2

CIRCLE (360, 166), 10, 2     'Rechte Pupille
PAINT (360, 166), 2, 2

LINE (320, 180)-(320, 240), 15 'Nase

LINE (260, 220)-(320, 280), 4 'Mund
LINE (320, 280)-(380, 220), 4
```

Lustiges Gesicht, nicht wahr?

Jetzt noch eine weitere Option, die man bei allen bisher besprochenen Grafiken einsetzen kann.

Sie heißt *STEP* und gibt an, daß die nächsten Bildschirmkoordinaten nicht absolut, sondern relativ zu den letzten Bildschirmkoordinaten sind. Positive Koordinaten bedeuten, daß sich der Bildpunkt um die angegebene Zahl von Punkten rechts unten vom letzten Bildpunkt befindet. Negative Koordinaten dagegen geben an, daß sich der neue Bildpunkt links oben von dem alten befindet.

Dadurch lassen sich Polygone (Vielecke) sehr einfach mit dem *LINE*-Befehl zeichnen, denn wenn *STEP* verwendet wird, kann der Startpunkt, von dem die Linie ausgeht, weggelassen werden.

Nun zum *DRAW*-Befehl. Er ist im Grunde genommen nichts anderes als der *PLAY*-Befehl, nur für Grafiken. Hier eine Liste mit den wichtigsten *DRAW*-Kommandos:

```
Ux..... Cursor um x Bildpunkte nach oben bewegen
Dx..... Cursor um x Bildpunkte nach unten bewegen
Lx..... Cursor um x Bildpunkte nach links bewegen
Rx..... Cursor um x Bildpunkte nach rechts bewegen
Hx..... Cursor um x Bildpunkte nach oben-links
Ex..... Cursor um x Bildpunkte nach oben-rechts
Gx..... Cursor um x Bildpunkte nach unten-links
Fx..... Cursor um x Bildpunkte nach unten-rechts

Mx,y..... Cursor wird nach x,y bewegt (absolute Koordinaten)
M+x,y..... Cursor wird nach x,y bewegt (relative Koordinaten)
M-x,y..... Cursor wird nach x,y bewegt (relative Koordinaten)
```

B..... Präfix: Cursor wird ohne zu zeichnen bewegt  
 N..... Präfix: Zeichnet und setzt Cursor auf Ausgangsposition zurück

Den Rest könnt ihr in der Online-Hilfe nachlesen.

## KAPITEL 8

### Datentypen in QBasic

Wenn man die Datentypen mit der Mathematik vergleicht, so entsprechen sie am ehesten den Zahlenmengen. Das heißt, daß die Datentypen sozusagen die Grundmengen für die Variablen festlegen.

Bis jetzt sind uns zwei Datentypen bekannt: Fließkommazahlen und Zeichenketten.

Der Datentyp der Fließkommazahlen, wie wir sie kennen, heißt *SINGLE*. Wie der Name sagt, hat er nur eine begrenzte Anzahl an Vor- und Nachkommastellen. Eine höhere Anzahl von Stellen hat der Datentyp *DOUBLE*.

Beide Fließkommazahlendatentypen haben jedoch den Nachteil, daß sie sehr langsam sind, weil der Computer intern keine Fließkommazahlen verarbeiten kann und sie deshalb vorher von QBasic umgewandelt werden müssen.

Deshalb gibt es mehrere Ganzzahl-Datentypen. Einer von ihnen, *INTEGER*, belegt 2 Byte Arbeitsspeicherplatz (pro Variable, versteht sich) und kann ganzzahlige Werte von -32768 bis +32767 annehmen. *LONG* belegt doppelt soviel Speicherplatz und kann  $-(2^{32})$  bis  $+(2^{32}-1)$  annehmen.

Der Zeichenkettendatentyp oder *STRING* besteht im Grunde genommen auch nur aus mehreren Integers, denn die einzelnen Zeichen werden als Zahlen, und zwar im sogenannten ASCII-Code, gespeichert. Wenn man es ganz genau nimmt, besteht ein String aus mehreren Bytes (Ganzzahlen von 0 bis 255, belegen jeweils ein Byte, gibt's nur in Power Basic).

Will man nun eine Variable mit einem bestimmten Datentyp anlegen, muß man den Befehl *DIM* anwenden. Seine Syntax lautet:

```
DIM Variable AS Datentyp
```

Wenn man eine Variable als String definiert, braucht man bei ihr nicht mehr das Dollarzeichen schreiben! Praktisch, oder?

Bei Strings gibt es noch eine Besonderheit. Normalerweise ist die Länge der Strings variabel. Der String kann dann eine Länge von 0 bis 32768 Zeichen haben. Mit *DIM* kann man aber auch Strings mit einer festen Länge definieren. Wird eine zu lange Zeichenkette zugewiesen, so werden die überflüssigen Zeichen einfach abgeschnitten. Wird eine zu kurze Zeichenkette zugewiesen, so wird der Rest mit Leerzeichen ausgefüllt. Beispiel:

```
'String-Demo
DIM Zeichenkette AS STRING * 8           'String mit fester Länge von 8 Zeichen
SCREEN 12                                'Bildschirm ausfüllen
PAINT (320, 200), 9
Zeichenkette = "Hallo! Wie geht's?"     'Erster Versuch
PRINT Zeichenkette
PRINT
Zeichenkette = "Hi!"                    'Zweiter Versuch
PRINT Zeichenkette
```

Mit dem *DIM*-Befehl lassen sich auch sogenannte Felder (englisch: Arrays) definieren. Die Anweisung

```
DIM FeldName(1 TO x) AS INTEGER
```

erzeugt ein Feld, das aus x Integer-Variablen besteht. Jede dieser Variablen läßt sich dann einzeln ansprechen, als ob sie nicht in einem Feld stünde. Will man beispielweise der fünften Variablen des Feldes *GanzZahl* den Wert 10 zuweisen, so schreibt man einfach:

```
GanzZahl(5) = 10
```

Damit das ganze verständlich wird, hier ein Beispielprogramm:

```
'Array-Demo

DIM Summand(1 TO 3) AS SINGLE
DIM Summe AS SINGLE
DIM Zaehler AS INTEGER

COLOR 14, 1
CLS

FOR Zaehler = 1 TO 3
  PRINT "Summand"; Zaehler;
  INPUT Summand(Zaehler)
  Summe = Summe + Summand(Zaehler)
NEXT

PRINT
PRINT Summand(1); "+"; Summand(2); "+"; Summand(3); "="; Summe
```

Da dieses Beispielprogramm etwas komplizierter ist als die anderen, wollen wir es Zeile für Zeile besprechen.

In der 3. Zeile wird ein Single-Feld mit drei Elementen namens *Summand* definiert.

In der 4. und der 5. Zeile werden zwei weitere Variablen definiert.

Anschließend werden die Farben gewählt und der Bildschirm gelöscht.

In der 10. Zeile steht nun der Anfang einer *FOR/NEXT*-Schleife. Zuerst wird ausgegeben, der wievielte *Summand* eingegeben werden soll. Danach muß der Benutzer diesen *Summand* eingeben.

Ihr merkt, daß man auch Variablen als "Arrayindex" verwenden kann. Bisher verwendeten wir dazu nur Zahlen.

Anschließend wird der eingegebene *Summand* gleich zu der *Summe* addiert. Man könnte zwar auch nachträglich alle drei *Summanden* zusammenzählen und der Variablen *Summe* zuweisen, doch das würde bei vielen *Summanden* sehr lange dauern.

Bei nur drei *Summanden*, wie wir in diesem Beispiel verwenden, ist der Geschwindigkeitsverlust zwar nicht bemerkbar, aber trotzdem sollten wir es nicht vernachlässigen, unsere Programme zu optimieren. (Eine Spezialität! Tatsache, daß in Winword viele Bugs enthalten sind...) Später wollen wir ja nicht nur Programme zum Addieren dreier *Summanden* schreiben, sondern auch solche Programme, mit denen man Space-Shuttles starten, steuern und landen lassen kann.

Nach der Schleife wird die Rechnung auf dem Bildschirm ausgegeben, aber das ist sowieso klar.

Wie wir bereits wissen, ist der Datentyp einer Variablen, wenn sie vorher nicht definiert wurde, automatisch *SINGLE*. Das können wir aber auch ändern. Wenn wir am Anfang eines Programms *DEFINT A-Z* schreiben, sind die nicht definierten Variablen automatisch vom Typ Integer. Es gibt auch *DEFLLNG* (Long), *DEFDBL* (Double), *DEFSTR* (String) und natürlich *DEFSNG* (Single). Wie sich wohl jeder selbst denken kann, ist *DEFSNG* Standard.

Lange keine Hausaufgaben mehr gehabt! Nun, mit den in den letzten Folgen erlernten Befehlen können wir unser Rechenprogramm aus Teil 3 ausbauen.

Verändert es so, daß

- die Zahlen in einem Array gespeichert werden,
- das Programm den Benutzer nach der Berechnung fragt, ob er zwei neue Zahlen eingeben will (und auch dementsprechend handelt),
- das ganze mit einigen Grafiken aufgelockert wird!

## Lösungen zu Kapitel 8

**Aufgabe 1:** Die Problemstellung war, unser Megarechenprogramm aus Folge 3 zu verbessern. Es gibt sicher eine Menge verschiedener Lösungen. Hier nun eine mögliche Lösung.

```
'DAS MEGARECHENPROGRAMM!!!!                                VERSION 5
'... aus The Real Adok's Way to QBASIC.

'+++ Variablenzuweisungen
DEFINT A-Z                                                'Variablen ohne Kennung sind Integer
DIM Zahl(1 TO 2) AS SINGLE                                'Array Zahl

blau = 1                                                    'Farbe Blau
gelb = 14                                                  'Farbe Gelb
weiss = 7                                                  'Farbe Wei
hell = 8                                                  'Zusatz für Helligkeit

'+++ Hauptprogramm
SCREEN 9                                                  'EGA-Grafik
GOSUB Titelbild                                          'Sprung nach Label Titelbild
GOSUB Titelmusik                                         'Sprung nach Label Titelmusik
a$ = INPUT$(1)                                           'Auf Tastendruck warten

Start:
COLOR , blau                                             'Blauer Hintergrund
CLS                                                       'Bildschirm löschen

COLOR hell + weiss                                       'Eingabe der ersten Zahl
LOCATE 2, 2
PRINT "Wie lautet die erste Zahl";
COLOR gelb
INPUT Zahl(1)

COLOR hell + weiss                                       'Eingabe der zweiten Zahl
LOCATE 5, 2
PRINT "Wie lautet die zweite Zahl";
COLOR gelb
INPUT Zahl(2)

CLS                                                       'Bildschirm löschen
COLOR hell + weiss                                       'Erste Zahl ausgeben
LOCATE 2, 2
PRINT "Zahl 1:";
COLOR gelb
PRINT Zahl(1);

COLOR hell + weiss                                       'Zweite Zahl ausgeben
LOCATE , 60
PRINT "Zahl 2:";
COLOR gelb
PRINT Zahl(2)

COLOR hell + weiss                                       'Verzierungsstrichlein ausgeben
LOCATE 3, 1
PRINT "-----";
PRINT "-----";
PRINT "-----";
PRINT "-----"

COLOR hell + weiss                                       'Ergebnis der Addition ausgeben
LOCATE 5, 2
```

```

PRINT "Addition:";
COLOR gelb
LOCATE , 17
PRINT Zahl(1) + Zahl(2)

COLOR hell + weiss
LOCATE 6, 2
PRINT "Subtraktion:";
COLOR gelb
LOCATE , 17
PRINT Zahl(1) - Zahl(2)

COLOR hell + weiss
LOCATE 7, 2
PRINT "Multiplikation:";
COLOR gelb
LOCATE , 17
PRINT Zahl(1) * Zahl(2)

COLOR hell + weiss
LOCATE 8, 2
PRINT "Division:";
COLOR gelb
LOCATE , 17
IF Zahl(2) = 0 THEN
  PRINT " Division durch Null!"
ELSE
  PRINT Zahl(1) / Zahl(2)
END IF

PLAY "t200 l8 o5 ddd p64 l2 g"

DO
  COLOR hell + weiss
  LOCATE 18, 2
  PRINT "Noch eine Berechnung (J/N)";
  COLOR gelb
  PRINT "?"
  Wahl$ = INPUT$(1)
LOOP UNTIL Wahl$ = "J" OR Wahl$ = "j" OR Wahl$ = "N" OR Wahl$ = "n"

SELECT CASE Wahl$
  CASE "J", "j"
    GOTO Start
  CASE ELSE
    END
END SELECT

'+++ Titelbild.
Titelbild:
LOCATE 3, 27
PRINT "DAS MEGARECHENPROGRAMM!!!!"
LOCATE 5, 21
PRINT "... aus The Real Adok's Way to QBASIC."

CIRCLE (320, 200), 100, 9
PAINT (320, 200), 9, 9

CIRCLE (280, 166), 30, 14
PAINT (280, 166), 14, 14

CIRCLE (360, 166), 30, 14
PAINT (360, 166), 14, 14

CIRCLE (280, 166), 10, 2
PAINT (280, 166), 2, 2

CIRCLE (360, 166), 10, 2
PAINT (360, 166), 2, 2

LINE (320, 180)-(320, 240), 15

```

```

LINE (260, 220)-(320, 260), 4      'Mund
LINE (320, 260)-(380, 220), 4      '
RETURN                               'Rückkehr zum Hauptprogramm

```

```

'+++ Titelmusik.
Titelmusik:
PLAY "t120 l8 o4"                   'Der Lärm fängt an!
PLAY "g g g > c c c < l4 a f c"     '
PLAY "l8 g g g > c c c l4 d < b g"  '
PLAY "l8 b- b- b- > e- e- e- l4 d < b- f"
PLAY "l8 e- e- e- a- a- a- l4 g > l8 e p8 < l4 c"
RETURN                               'Rückkehr zum Hauptprogramm

```

```
'+++ Ende des Programms.
```

Wie ihr seht, habe ich mir sehr viel Mühe gemacht, den Code übersichtlich zu gestalten. Ich hoffe, daß sich meine Mühe gelohnt hat.

Dieses Programm werden wir in diesem Basic-Kurs nur noch ein einziges Mal verbessern, und zwar mit den Befehlen, die wir in Folge 11 besprechen werden.

## KAPITEL 9

### Dateiverwaltung

Oft muß man DOS-Befehle oder externe Programme aus einem eigenen Programm heraus ausführen. Der Befehl hierzu ist *SHELL*. Mit ihm kann man sich eine Kommandozeile basteln. Diese wird dann 1:1 an DOS übergeben. Für den, der sich mit DOS auskennt, ist der *SHELL*-Befehl also überhaupt kein Problem. Die Kommandozeile muß als String übergeben werden.

```
SHELL "DIR /O"
```

zeigt beispielsweise den Inhalt des aktuellen Verzeichnisses auf dem Bildschirm alphabetisch sortiert an. Auch solche Aufrufe sind möglich:

```
SHELL Pfad$ + DateiName$
```

QBasic sieht zunächst nach, wie die Inhalte der Variablen *Pfad\$* und *DateiName\$* aussehen. Danach werden die beiden Zeichenketten zu einer zusammengefügt.

Steht z.B. in *Pfad\$* "C:\WINDOWS\I" und in *DateiName\$* "WIN.COM", so wird die Zeichenkette "C:\WINDOWS\IWIN.COM" übergeben. Das Pluszeichen dient bei Strings also dazu, die beiden Zeichenketten aneinanderzuhängen. Dies kann oft sehr nützlich sein und funktioniert bei allen Befehlen und Funktionen, in denen Strings verwendet werden.

```
PRINT Pfad$ + DateiName$
```

wäre zwar auch möglich, aber dazu gibt es ja schon den Strichpunkt:

```
PRINT Pfad$; DateiName$
```

Diesen Strichpunkt gibt es bei *SHELL* nicht, weshalb man das Pluszeichen benötigt.

Außerdem kann man ja auch zwei verkettete Strings einer anderen Stringvariablen zuweisen. Das geht dann genauso, wie wenn man zwei Zahlen addiert und diese einer Variablen zuweist. Nur bei *LINE INPUT* sollte man Strings nicht verketteten...!

Zurück zu *SHELL*. Fassen wir zusammen: Der *SHELL* als Parameter übergebene String wird dann von MS-DOS so ausgeführt, als ob ihn der Anwender in der Kommandozeile eingegeben hätte. Der *SHELL*-Befehl wird

in Spielen benutzt, um Bilder, die mit dem Graphic Workshop zu selbstlaufenden EXE-Dateien umgewandelt wurden, anzuzeigen.

Ein Beispielprogramm wäre hier sinnlos, weil es ja nichts machen würde, außer wieder ein anderes Programm zu starten.

Kommen wir lieber zur eigentlichen Dateiverwaltung. Der wichtigste Befehl heißt hier wohl *OPEN*. Mit ihm läßt sich eine Datei erzeugen oder einlesen. Seine Syntax lautet:

```
OPEN Datei FOR Modus AS Nummer
```

*Datei* ist ein String, der den Dateinamen enthält. Auf *Modus* kommen wir noch später zurück. *Nummer* ist eine Zahl, unter der die Dateiverwaltungsbefehle die Datei ansprechen können.

Vor dem Ende eines jeden Programms müssen alle geöffneten Dateien geschlossen werden. Dazu gibt es den Befehl *CLOSE*. Seine Syntax lautet:

```
CLOSE Nummer
```

So, kommen wir nun zum Modus.

In diesem Basic-Kurs werden wir die zwei Modi *INPUT* und *OUTPUT* kennenlernen. Wer etwas über die anderen Modi erfahren will, soll in der Online-Hilfe oder in einem QBasic-Buch nachlesen.

Der Modus (nicht Befehl!) *INPUT* dient dazu, eine Datei zeilenweise einzulesen. Wie der Name sagt, funktioniert dies mit einer speziellen Unterfunktion von *LINE INPUT*. Die Syntax lautet:

```
LINE INPUT #Nummer, Variable
```

*Nummer* ist die Nummer der Datei, aus der eine Zeile gelesen, und *Variable* eine Stringvariable, in der die Zeile gespeichert werden soll. Beim ersten Aufruf wird die erste Zeile gelesen, beim zweiten die zweite usw.

Dateien lassen sich auch mit *INPUT* und *INPUT\$* einlesen. Die Syntax von letzterem lautet:

```
Variable = INPUT$(AnzahlZeichen, Nummer)
```

Hier wird die Datei Zeichen für Zeichen eingelesen.

Wichtig ist auch die Funktion *EOF*. Wenn *EOF(1)* einen Wert ungleich 0 ergibt, dann ist das Dateiende der Datei mit der Nummer 1 erreicht. Wird jetzt noch einmal ein Lesezugriff mit den oben genannten Befehlen getätigt, wird das Programm mit einer Fehlermeldung abgebrochen.

Das muß man natürlich verhindern, indem man, wenn *EOF(1)* ungleich ist, das Einlesen der Zeilen abbricht. Hier gleich ein praktisches Anwendungsbeispiel, nämlich ein File-Viewer.

```
'File-Viewer, unbrauchbare Version
```

```
COLOR 15, 1
CLS
```

```
PRINT "Dateiname";           'Name der Datei abfragen
COLOR 14
PRINT "? ";
LINE INPUT Datei$
```

```
OPEN Datei$ FOR INPUT AS 1   'Datei öffnen
```

```
CLS
COLOR 15
```

```
'Zeilen einlesen und anzeigen
DO UNTIL EOF(1)              'Wenn kein Vergleichoperator vorhanden ist,
  LINE INPUT #1, Zeile$      'wird automatisch > 0 ergänzt!
  PRINT Zeile$
LOOP
```

```
CLOSE 1 'Datei schließen
```

Warum ich 'unbrauchbare Version' geschrieben habe? Ganz einfach: Probiert doch einmal aus, eine große Datei zu öffnen (z.B. KURS0019.BAS)!

Es tritt der gleiche Effekt wie beim DOS-Befehl *TYPE* ohne *MORE* auf: Die Datei wird 'in einem durch' angezeigt. Effektiv läßt sich dann nur die letzte Seite betrachten.

*TYPE* kann man ja durch *PAUSE* pausieren, doch das geht bei unserem Viewer nicht! Am besten, wir bauen gleich eine seitenweise Anzeige wie bei *MORE* ein. Wenn der Benutzer dann die nächste Seite sehen will, betätigt er einfach eine beliebige Taste.

Das können wir erreichen, indem wir einen Zähler einbauen, in dem gespeichert wird, wieviele Zeilen schon ausgegeben wurden. Mit *IF/THEN/ELSE/ENDIF* wird dann abgefragt, ob er einen bestimmten Wert (sagen wir, 20) erreicht hat. Wenn ja, muß der Benutzer eine beliebige Taste drücken.

Danach wird der Bildschirm gelöscht, der Zähler auf 0 gesetzt, und weiter geht's!

Hier das Listing. .... Wieso denn? Es wäre doch eine gute Idee, euch die Umsetzung in Basic als Hausaufgabe aufzugeben! Den Programmablauf haben wir ja jetzt schon besprochen.

Kommen wir zum nächsten Modus, *OUTPUT*. Dieser Modus ist das genaue Gegenteil des *INPUT*-Modi, denn hier wird eine neue Datei erzeugt und dann in diese geschrieben.

Der Schreibbefehl ist auch *PRINT*, nur muß wie bei *LINE INPUT* die Dateinummer angegeben werden. Als Beispiel ein einfacher ASCII-Texteditor.

```
'Texteditor
COLOR 15, 1
CLS

'Eingabe des Dateinamens
PRINT "Gib den Namen der Datei ein!"
PRINT "Vorsicht: Falls eine Datei mit demselben Namen bereits vorhanden ist,"
PRINT "wird sie überschrieben!"
COLOR 14
LINE INPUT Datei$

CLS
COLOR 15
PRINT "Jetzt Text eingeben! Um das Programm zu beenden, Leerzeile eingeben."
COLOR 14

'Eingabe des Texts
OPEN Datei$ FOR OUTPUT AS 1
DO
  LINE INPUT Zeile$
  PRINT #1, Zeile$
LOOP UNTIL Zeile$ = ""
CLOSE 1
```

Das war für heute alles!

Eine Hausaufgabe habt ihr ja schon bekommen, aber wie immer sollt ihr euch auch mit den neuen Befehlen herumspielen. Nur Vorsicht bei *OUTPUT*! Wenn schon eine Datei mit dem gewählten Namen existiert, wird sie gnadenlos überschrieben!

## Lösungen zu Kapitel 9

**Aufgabe 1:** Ihr müßtet bei unserem File-Viewer seitenweises Anzeigen der Datei ergänzen. Eine mögliche Lösung wäre folgendes Programm:

```
'File-Viewer, brauchbare Version

DEFINT A-T

COLOR 15, 1
CLS

PRINT "Dateiname";           'Name der Datei abfragen
COLOR 14
PRINT "? ";
LINE INPUT Datei$

OPEN Datei$ FOR INPUT AS 1    'Datei öffnen

CLS
COLOR 15

'Zeilen einlesen und anzeigen
Zaehler = 0
DO UNTIL EOF(1)               'Wenn kein Vergleichoperator vorhanden ist,
  LINE INPUT #1, Zeile$      'wird automatisch > 0 ergänzt!
  PRINT Zeile$
  Zaehler = Zaehler + 1
  IF Zaehler = 20 THEN
    a$ = INPUT$(1)
    CLS
    Zaehler = 0
  END IF
LOOP

CLOSE 1                       'Datei schließen
```

## KAPITEL 10

### Befehle und Funktionen

Heute geht es um *SUBs* und *FUNCTIONs*, zwei der wichtigsten Merkmale der strukturierten Programmierung. Bevor wir damit anfangen können, müssen wir zunächst den Unterschied zwischen Befehlen und Funktionen klären.

Befehle wie *PRINT* veranlassen den Computer, etwas Bestimmtes zu tun, z.B. einen Text auszugeben. Es lassen sich Parameter übergeben, die mehrere Optionen möglich werden lassen.

An Funktionen wie *INPUT\$* lassen sich auch Parameter übergeben. Doch Funktionen sind vielseitig. Eine Funktion gibt nämlich im Gegensatz zu einem Befehl einen Wert zurück. Mit diesem Wert kann man nun machen, was man will: ihn einer Variablen zuweisen, auf dem Bildschirm ausgeben, testen usw. Folgende Anwendungen sind gültig:

```
a$ = INPUT$(1)                'der Variablen a$ wird das Ergebnis der Funktion
                              'INPUT$(1) zugewiesen
PRINT INPUT$(1)              'das Ergebnis der Funktion INPUT$(1), also die
                              'gedrückte Taste, wird auf dem Bildschirm
                              'ausgegeben
IF INPUT$(1) = "J" THEN...    'das Ergebnis der Funktion INPUT$(1) wird mit
                              '"J" verglichen
```

Es gibt natürlich auch unzählige andere Anwendungen, wie z.B. *SHELL INPUT\$(1)* (sehr sinnig), aber diese sollten das Prinzip klar machen.

Mit den Befehlen *SUB* und *FUNCTION* kann man nun seine eigenen Befehle bzw. Funktionen erstellen. Natürlich sind es wohlgerneht nicht ganz neue Befehle, sondern nur eine Zusammenfassung von Befehlen.

Man spricht von einem *Programm im Programm* bzw. *Unterprogramm*. Dieser Ausdruck stammt zwar aus der Zeit, in der GW-Basic und Basic 2.0 supertolle Hits waren, aber er wird trotzdem heute immer noch verwendet.

Erstellen wir doch einfach 'mal eine einfache *SUB*, um zu sehen, wie diese funktionieren!

```
'SUB-Demo
DECLARE SUB Hallo ()

DEFINT A-Z
'Beispielprogramm
FOR i = 1 TO 10
  Hallo
NEXT

DEFINT A-Z
'Eigentliche SUB
SUB Hallo
  PRINT "Hallo, Welt!"          'Ein bißchen spät, nicht wahr?
END SUB
```

Jetzt werde ich viel erklären müssen.

Nach Eingabe der Zeile *SUB Hallo* eröffnet QBasic ein eigenes Fenster für die *SUB*. Mit *F2* kann man zwischen den Fenstern, in denen sich die *SUBs* und *FUNCTIONs* befinden, und dem Hauptprogramm hin- und herschalten.

Gehen wir das Programm nun Zeile für Zeile durch.

Wozu Zeile 2 (*DECLARE SUB Hallo*) gut ist, müßt ihr nicht wissen. Sie wird von QBasic automatisch eingefügt.

In der vierten Zeile werden mit *DEFINT A-Z* alle Variablen, die nicht definiert sind, auf den Datentyp *INTEGER* gesetzt.

Was ich euch in Folge 7 noch nicht sagen konnte, weil wir die *SUBs* und *FUNCTIONs* nicht kannten, ist, daß dieser Aufruf nur für das Hauptprogramm gilt. Für die *SUBs* und die *FUNCTIONs* müssen wir *DEFINT A-Z* extra schreiben. Wir können für sie aber auch unabhängig vom Hauptprogramm *DEFSNG*, *DEFLNG* etc. verwenden.

Mit den Zeilen 6 bis 8 wird nun erreicht, daß die *SUB Hallo* zehnmal aufgerufen wird.

In QBasic und Quick Basic kann man *SUBs* aufrufen, als wären sie normale Befehle. Wir haben beispielsweise nur *Hallo* schreiben müssen.

In Power Basic muß man vor jedem Aufruf einer *SUB* oder einer *FUNCTION* den Befehl *CALL* setzen.

Ab Zeile 10 beginnt nun die eigentliche *SUB*. Zuerst wird mit *DEFINT A-Z* festgelegt, daß auch sie, wenn nichts anderes angegeben wird, mit Integerwerten arbeitet.

In Zeile 12 befindet sich nun der eigentliche Kern der *SUB*: der *SUB*-Befehl. Hinter dem Schlüsselwort *SUB* muß der Name der *SUB* stehen. In unserem Beispiel lautet dieser *Hallo*. Die Befehle, die nun beim Aufruf dieser *SUB* ausgeführt werden, stehen zwischen der Zeile mit dem Befehl *SUB* und der Zeile *END SUB*.

Im Grunde genommen sind *SUBs* also nichts anderes als Blöcke, die man an einer beliebigen Stelle des Hauptprogramms oder einer anderen *SUB/FUNCTION* aufrufen kann. Im Extremfall kann eine *SUB* sogar ein eigenes Programm beinhalten!

Fassen wir also zusammen: Eine *SUB* ist ein Block von Anweisungen, der an jeder beliebigen Stelle des Hauptprogramms oder einer anderen *SUB/FUNCTION* aufgerufen werden kann. Er beginnt mit dem Befehl *SUB* und endet mit *END SUB*. Dazwischen befinden sich die eigentlichen Befehle.

Neben dem Befehl *SUB* steht der Name dieser *SUB*. Eine *SUB* wird mit ihrem Namen aufgerufen. In Power Basic muß vorher der Befehl *CALL* stehen.

*SUBs* wären nicht *SUBs*, wenn man sie nicht mit Parametern aufrufen könnte. Damit eine *SUB* mit Parametern aufgerufen werden kann, müssen diese zunächst in der Zeile mit dem Befehl *SUB* neben dem Namen der *SUB* definiert werden. Beispiel:

```
SUB DruckeText (Text AS STRING, Anzahl AS INTEGER)
```

Bevor hier jemand in Panik ausbricht, die Erklärung: Wenn die *SUB DruckeText* aufgerufen wird, müssen jetzt zwei Parameter übergeben werden. Wir haben mit obiger Zeile definiert, daß der erste vom Typ *STRING* und der zweite vom Typ *INTEGER* sein muß.

*Text* und *Anzahl* sind Variablennamen, unter denen die *SUB* die beiden Parameter ansprechen kann. Rufen wir jetzt die *SUB DruckeText* beispielsweise mit

```
CALL DruckeText ("Hallo!", 1000)
```

auf, so erhält die Variable *Text* den Wert "*Hallo!*" und *Anzahl* den Wert *1000*. Diese Variablen lassen sich jedoch nur in der *SUB* ansprechen, wie wir später erfahren werden.

Wird eine *SUB* mit Parametern aufgerufen, ist *CALL* übrigens in allen drei Basic-Dialekten notwendig. Damit es jeder kapiert, ein Beispielprogramm:

```
'Parameter-Demo
DECLARE SUB DruckeText (Text AS STRING, Anzahl AS INTEGER)

DEFINT A-Z
'Beispielprogramm
COLOR 15, 1
CLS
CALL DruckeText("QBasic ist super!", 20)

DEFINT A-Z
'Eigentliche SUB
SUB DruckeText (Text AS STRING, Anzahl AS INTEGER)
  FOR i = 1 TO Anzahl
    PRINT Text
  NEXT
END SUB
```

Falls jemand noch immer Probleme mit den Parametern haben sollte: einfach fragen!

Kommen wir nun zu einem weiteren wichtigen Merkmal von *SUBs* und *FUNCTIONs*, den lokalen Variablen.

Vorhin schrieb ich, daß sich die Variablen *Text* und *Anzahl* nur in der eigentlichen *SUB* ansprechen lassen. Die Frage ist: Warum?

Beginnen wir ganz langsam: Auch alle Variablen, die im Hauptprogramm verwendet werden, lassen sich nur im Hauptprogramm ansprechen. Soll eine *SUB* oder eine *FUNCTION* eine Variable des Hauptprogramms ansprechen können, so muß diese als Parameter übergeben werden.

Ebenso können das Hauptprogramm oder andere *SUBs/FUNCTIONs* nicht die Variablen, die in dieser *SUB* oder *FUNCTION* verwendet werden, ansprechen.

Das bedeutet schlicht und einfach: Wenn in einer *SUB/FUNCTION* eine Variable verwendet wird und in einer anderen *SUB/FUNCTION* oder dem Hauptprogramm eine Variable mit demselben Namen verwendet wird, so sind das zwei verschiedene Variablen!

Am besten, wir lassen das von einem Beispielprogramm demonstrieren.

```
DECLARE SUB ZeigeText ()
'Lokale Variablen

COLOR , 1
CLS
```

```

Text$ = "Tschüß!"

ZeigeText

COLOR 15
PRINT "Wir befinden uns im Hauptprogramm. Der Inhalt von Text$ ist: ";
COLOR 14
PRINT Text$

SUB ZeigeText
  Text$ = "Hallo!"
  COLOR 15
  PRINT "Wir befinden uns in der SUB ZeigeText. Der Inhalt von Text$ ist: ";
  COLOR 14
  PRINT Text$
END SUB

```

Wozu sind solche lokalen Variablen gut? Nun, sie haben folgende Vorteile:

- Jede *SUB* oder *FUNCTION* hat ihre eigenen lokalen Variablen. Auch, wenn diese dieselben Namen wie die Variablen aus einer anderen *SUB/FUNCTION* oder dem Hauptprogramm haben, sind sie 'eigenständige' Variablen.
- Nach dem Beenden der *SUB/FUNCTION* werden alle lokalen Variablen automatisch gelöscht. Gebe es keine lokalen Variablen, so müßte dies der Programmierer tun.

Daß die lokalen Variablen gelöscht werden, kann man übrigens dadurch verhindern, daß man am Schluß der Zeile mit *SUB* das Schlüsselwort *STATIC* schreibt.

Es gibt außerdem noch globale Variablen. Diese können in allen *SUBs*, *FUNCTIONs* und im Hauptprogramm angesprochen werden.

Um eine globale Variable zu erzeugen, muß man sie auf alle Fälle mit *DIM* definieren. Zwischen *DIM* und dem Namen der Variablen muß man aber noch *SHARED* schreiben.

Jetzt wissen wir im großen und ganzen, wie man *SUBs* erzeugt und aufruft. Nun stellt sich die Frage, wozu solche *SUBs* überhaupt gut sind. Immerhin lassen sich Unterprogramme auch mit *GOSUB* aufrufen! Die Verwendung von *SUBs* bringt aber folgende Vorteile:

- Lokale Variablen (siehe oben).
- Übersichtlichkeit. Zumindestens in QBasic und Quick Basic kann man mit *F2* eine Liste aller *SUBs* aufrufen und komfortabel zwischen ihnen wechseln.
- Die *SUBs* sind vom Hauptprogramm genau abgegrenzt, während Unterprogramme, die mit *GOSUB* aufgerufen werden, Teile des Hauptprogramms sind. Deshalb ist es wichtig, vor die *GOSUB*-Unterprogramme *END* zu setzen. Bei *SUBs* ist dies nicht notwendig.

Außerdem haben *SUBs* noch einen Vorteil, den alle Blöcke besitzen (und den ich euch schon längst hatte sagen müssen).

Es gibt einen Befehl namens *EXIT*, mit dem man aus einem Block "aussteigen" kann. Als Parameter muß der Name des Blocks, aus dem ausgestiegen werden soll, angegeben werden, also: *EXIT IF*, *EXIT DO*, *EXIT FOR*, *EXIT SUB*, *EXIT FUNCTION*,...

Als Beispiel hier nochmals die Zählschleife mit *DO/LOOP* aus Folge 4, diesmal unter Verwendung des *EXIT*-Befehls.

```

'Zählschleife

COLOR 14, 1
CLS

Zaehler = 0
DO
  Zaehler = Zaehler + 1

```

```

PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."

IF Zaehler = 10 THEN                                'Wenn Schleifenzähler 10 erreicht hat,
  EXIT DO                                           'wird aus der DO-Schleife
END IF                                              'herausgesprungen.
                                                    '
                                                    ' |||
LOOP                                               ' hierher wird gesprungen
                                                    ' \ \
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."

```

Ihr habt recht, wenn ihr jetzt sagt: "Das geht ja auch mit *GOTO*!" Aber diese Methode ist weder elegant noch speicherplatzsparend (ein zusätzliches Label muß definiert werden). Damit ihr dies selbst seht, hier die Version mit *GOTO*:

```

'Zählschleife

COLOR 14, 1
CLS

Zaehler = 0
DO
  Zaehler = Zaehler + 1
  PRINT "Schleife wird zum"; Zaehler; ". Mal durchlaufen."

  IF Zaehler = 10 THEN                                'Wenn Schleifenzähler 10 erreicht hat,
    GOTO Label                                       'wird zu 'Label' gesprungen.
  END IF

LOOP

Label:                                               'hierher wird gesprungen
PRINT "Wert des Schleifenzählers ist"; Zaehler; "=> PROGRAMMABBRUCH."

```

Das war übrigens unser 35. Beispielprogramm! Super, nicht wahr?

Jetzt kennen wir uns mit den *SUBs* bestens aus (sollten wir zumindest). Nun kommen die *FUNCTIONs* an die Reihe. Sie sind im Grunde genommen bis auf folgende Unterschiede mit den *SUBs* identisch:

- In der Anfangszeile wird statt *SUB FUNCTION* geschrieben.
- Ebenso in der Endzeile.
- Am Ende des Namens der *FUNCTION* steht ein Zeichen, das angibt, welchen Datentyp der Rückgabewert besitzt. Folgende Zeichen sind zulässig:  
 %...Integer,  
 &...Long,  
 !...Single,  
 #...Double,  
 \$...String.
- Der Rückgabewert muß vor dem Ende der *FUNCTION* in einer Variablen gespeichert werden, die genauso wie die *FUNCTION* heißt (inklusive des letzten Zeichens).
- *FUNCTIONs* werden nicht wie Befehle, sondern wie Funktionen aufgerufen.

Als Beispiel habe ich ein Programm geschrieben, mit dem man die n-te Wurzel einer Zahl ausrechnen kann. Ich verwendete dazu den Operator '^' (hoch).

```

'FUNCTION-Demo
DECLARE FUNCTION nteWurzel# (Zahl AS DOUBLE, n AS DOUBLE)

DEFDBL A-Z

COLOR 15, 1
CLS

PRINT "Von welcher Zahl soll die n-te Wurzel ausgerechnet werden";
COLOR 14
INPUT Zahl

PRINT

```

```

COLOR 15
PRINT "Wieviel ist n";
COLOR 14
INPUT n

PRINT
COLOR 15
IF n = 0 THEN
  PRINT "Die 0-te Wurzel kann nicht ausgerechnet werden!"
ELSE
  PRINT "Die n-te Wurzel der Zahl ist";
  COLOR 14
  PRINT nteWurzel#(Zahl, n);
  COLOR 15
  PRINT "."
END IF

DEFDBL A-Z
FUNCTION nteWurzel# (Zahl AS DOUBLE, n AS DOUBLE)
  nteWurzel# = Zahl ^ (1 / n)
END FUNCTION

```

Die mathematische Formel möchte ich euch nicht erklären (müßte in jedem Mathe-Lexikon stehen), es sei aber gesagt, daß sie dank den *DOUBLE*-Variablen mit einer sehr hohen Rechengenauigkeit von 15 (!) Nachkommastellen rechnet.

Die Funktion *nteWurzel#* ist übrigens Bestandteil der Toolbox, die im Kapitel 12 veröffentlicht wird.

## KAPITEL 11

### Funktionen zur Verarbeitung von Strings

Ich möchte euch jetzt eine Reihe von Funktionen zur Stringverarbeitung vorstellen.

**Funktion *ASC*.** Syntax: *Variable = ASC(Zeichen)*. Das Ergebnis ist der ASCII-Code des angegebenen Zeichens. In der Online-Hilfe sind alle ASCII-Codes zusammengefaßt. Beispiel:

```

'ASC-Demo

DIM Zeichen AS STRING * 1

COLOR 15, 1
CLS

PRINT "Gib ein beliebiges Zeichen ein!"
Zeichen = INPUT$(1)

PRINT
PRINT "Du hast das Zeichen ";
COLOR 14
PRINT Zeichen;
COLOR 15
PRINT " eingegeben. Sein ASCII-Code lautet";
COLOR 14
PRINT ASC(Zeichen);
COLOR 15
PRINT "."

```

**Funktion *CHR\$*.** Syntax: *Stringvariable = CHR\$(ASCIICode)*. Das Ergebnis ist das Zeichen mit dem angegebenen ASCII-Code. In der Online-Hilfe sind alle ASCII-Codes zusammengefaßt. Beispiel:

```

'CHR$-Demo

DIM ASCIICode AS INTEGER 'Wer Power Basic verwendet, kann hier auch BYTE
                          'schreiben.

```

```

COLOR , 1

DO
  CLS
  COLOR 15
  PRINT "Gib eine Zahl von 0 bis 255 ein!"
  COLOR 14
  INPUT ASCIICode
LOOP UNTIL ASCIICode >= 0 AND ASCIICode <= 255

COLOR 15
PRINT "Das Zeichen mit diesem ASCII-Code ist ";
COLOR 14
PRINT CHR$(ASCIICode);
COLOR 15
PRINT "."

```

*CHR\$(7)* ist insofern eine Ausnahme, als daß es kein Zeichen, sondern einen Piepton erzeugt (einfach ausprobieren!).

**Funktion *STRING\$*.** Syntax: *Stringvariable = STRING\$(Anzahl, Zeichen)*. Das Ergebnis ist eine Zeichenkette, die aus der angegebenen Anzahl des Zeichens besteht. Beispiel:

```

'Für Verliebte

COLOR 13, 0
CLS

FOR i = 1 TO 24
  PRINT STRING$(80, CHR$(3))
NEXT
PRINT STRING$(80, CHR$(3));
a$ = INPUT$(1)

```

**Funktion *MID\$*.** Syntax: *Stringvariable = MID\$(Zeichenkette, x, y)*. Das Ergebnis ist ein Teilstring der angegebenen Zeichenkette. Er beginnt mit dem *x*-ten Zeichen der ursprünglichen Zeichenkette und ist *y* Zeichen lang. *y* kann auch weggelassen werden. Dann reicht der Teilstring bis zum Ende der Zeichenkette. Da sich dies komplizierter anhört, als es in Wirklichkeit ist, hier ein Beispiel:

```

'MID$-Demo

COLOR 15, 1
CLS

PRINT "Gib eine Zeichenkette ein!"
COLOR 14
LINE INPUT Zeichenkette$

PRINT
COLOR 15
PRINT "Die ersten sechs Zeichen der eingegebenen Zeichenkette lauten ";
COLOR 14
PRINT MID$(Zeichenkette$, 1, 6);
COLOR 15
PRINT "."

PRINT
PRINT "Die Zeichen 2 bis 4 lauten ";
COLOR 14
PRINT MID$(Zeichenkette$, 2, 3);
COLOR 15
PRINT "."

```

Siehe dazu auch: *LEFT\$*, *RIGHT\$* (Online-Hilfe).

**Funktion *LTRIM\$*.** Syntax: *Stringvariable = LTRIM\$(Zeichenkette)*. Das Ergebnis ist ein String, der der ursprünglichen Zeichenkette gleicht, nur ohne überflüssige Leerzeichen vor dem eigentlichen Text. Siehe dazu auch: *RTRIM\$* (Online-Hilfe).

**Funktion LEN.** Syntax: *Variable = LEN(Zeichenkette)*. Das Ergebnis ist die Länge der Zeichenkette.

**Funktion UCASE\$.** Syntax: *Stringvariable = UCASE\$(Zeichenkette)*. Das Ergebnis gleicht der ursprünglichen Zeichenkette, nur die Kleinbuchstaben sind in Großbuchstaben umgewandelt worden. Siehe auch: *LCASE\$* (Online-Hilfe).

**Funktion STR\$.** Syntax: *Stringvariable = STR\$(Variable)*. Diese Funktion wandelt eine Zahlenvariable in einen String um. Dieser String kann dann wie alle anderen Strings bearbeitet werden. Beispiel:

```
'STR$-Demo

COLOR 15, 1
CLS

PRINT
PRINT "          1000 + 1000 = ";
COLOR 14
PRINT 1000 + 1000

COLOR 15
PRINT " STR$(1000) + STR$(1000) = ";
COLOR 14
PRINT STR$(1000) + STR$(1000)
```

Wozu ist die *STR\$*-Funktion sonst noch gut?

Wenn wir bisher Zahlen auf dem Bildschirm ausgaben, wurde vor und nach der Zahl jeweils ein Leerzeichen ausgegeben. Wenn man *STR\$* in Kombination mit *LTRIM\$* und *RTRIM\$* verwendet, kann man dieses Problem umgehen, wie nachfolgendes Programm zeigt:

```
'Ausgabe von Zahlenvariablen

COLOR 10, 1
CLS

PRINT "Direkte Ausgabe"
COLOR 15
PRINT "Die Zahl lautet ";
COLOR 14
PRINT 1000;
COLOR 15
PRINT "."
PRINT

COLOR 10
PRINT "Mit STR$, LTRIM$ und RTRIM$"
COLOR 15
PRINT "Die Zahl lautet ";
COLOR 14
PRINT LTRIM$(RTRIM$(STR$(1000)));
COLOR 15
PRINT "."
```

*VAL* ist das genaue Gegenteil von *STR\$*. Mit ihm kann man eine Zahlenvariable in einen String umwandeln.

Dies hat folgenden Vorteil: Habt ihr schon probiert, was passiert, wenn ihr in unserem Megarechenprogramm statt einer Zahl eine Zeichenkette eingibt? Um diese dumme Fehlermeldung zu umgehen, kann man den *INPUT*-Befehl durch einen *LINE INPUT*-Befehl ersetzen und anschließend die eingegebene Zeichenkette in eine Zahl umwandeln.

Mit den neuen Befehlen habe ich das Megarechenprogramm verbessert. Hier ist nun die neueste Version.

```
'DAS MEGARECHENPROGRAMM!!!!                               VERSION 6
'... aus The Real Adok's Way to QBASIC.
```

```

DECLARE SUB Titelbild ()           'Deklaration der SUB Titelbild
DECLARE SUB Titelmusik ()         'Deklaration der SUB Titelmusik

'+++ Variablenzuweisungen
DEFINT A-Z                       'Variablen ohne Kennung sind Integer
DIM Zahl(1 TO 2) AS SINGLE       'Array Zahl

blau = 1                          'Farbe Blau
gelb = 14                         'Farbe Gelb
weiss = 7                         'Farbe Weiß
hell = 8                          'Zusatz für Helligkeit

'+++ Hauptprogramm
SCREEN 9                          'EGA-Grafik
Titelbild                         'Aufruf der SUB Titelbild
Titelmusik                       'Aufruf der SUB Titelmusik
a$ = INPUT$(1)                   'Auf Tastendruck warten

Start:
COLOR , blau                     'Blauer Hintergrund
CLS                               'Bildschirm löschen

DO                                'Eingabe der ersten Zahl
  COLOR hell + weiss             '
  LOCATE 2, 2                   '
  PRINT "Wie lautet die erste Zahl"; '
  COLOR gelb                    '
  PRINT "? ";                   '
  LINE INPUT Temp$              '
  Temp$ = LTRIM$(RTRIM$(Temp$)) '
  Zahl(1) = VAL(Temp$)          '
LOOP UNTIL Temp$ = LTRIM$(RTRIM$(STR$(Zahl(1))))

DO                                'Eingabe der zweiten Zahl
  COLOR hell + weiss             '
  LOCATE 5, 2                   '
  PRINT "Wie lautet die zweite Zahl"; '
  COLOR gelb                    '
  PRINT "? ";                   '
  LINE INPUT Temp$              '
  Temp$ = LTRIM$(RTRIM$(Temp$)) '
  Zahl(2) = VAL(Temp$)          '
LOOP UNTIL Temp$ = LTRIM$(RTRIM$(STR$(Zahl(2))))

CLS                               'Bildschirm löschen
COLOR hell + weiss              'Erste Zahl ausgeben
LOCATE 2, 2                     '
PRINT "Zahl 1:";                '
COLOR gelb                      '
PRINT Zahl(1);                  '

COLOR hell + weiss              'Zweite Zahl ausgeben
LOCATE , 60                     '
PRINT "Zahl 2:";                '
COLOR gelb                      '
PRINT Zahl(2)                   '

COLOR hell + weiss              'Verzierungsstrichlein ausgeben
LOCATE 3, 1                     '
PRINT "-----";               '
PRINT "-----";               '
PRINT "-----";               '
PRINT "-----"                 '

COLOR hell + weiss              'Ergebnis der Addition ausgeben
LOCATE 5, 2                     '
PRINT "Addition:";              '
COLOR gelb                      '
LOCATE , 17                     '
PRINT Zahl(1) + Zahl(2)         '

```

```

COLOR hell + weiss                                'Ergebnis der Subtraktion ausgeben
LOCATE 6, 2                                        '
PRINT "Subtraktion:";                             '
COLOR gelb                                         '
LOCATE , 17                                        '
PRINT Zahl(1) - Zahl(2)                           '

COLOR hell + weiss                                'Ergebnis der Multiplikation ausgeben
LOCATE 7, 2                                        '
PRINT "Multiplikation:";                          '
COLOR gelb                                         '
LOCATE , 17                                        '
PRINT Zahl(1) * Zahl(2)                           '

COLOR hell + weiss                                'Ergebnis der Division ausgeben
LOCATE 8, 2                                        '
PRINT "Division:";                                '
COLOR gelb                                         '
LOCATE , 17                                        '
IF Zahl(2) = 0 THEN                                '
  PRINT " Division durch Null!"                   '
ELSE                                               '
  PRINT Zahl(1) / Zahl(2)                         '
END IF                                             '

PLAY "t200 l8 o5 ddd p64 l2 g"                   'Musik

DO                                                 'Abfrage, ob noch eine Berechnung
  COLOR hell + weiss                               '
  LOCATE 18, 2                                     '
  PRINT "Noch eine Berechnung (J/N)";             '
  COLOR gelb                                       '
  PRINT "?"                                        '
  Wahl$ = UCASE$(INPUT$(1))                        '
LOOP UNTIL Wahl$ = "J" OR Wahl$ = "N"            '

SELECT CASE Wahl$                                  'Auswertung der Wahl
  CASE "J"                                         '
    GOTO Start                                     '
  CASE ELSE                                        '
    END                                           '
END SELECT                                        '

'+++ Ende des Programms.

SUB Titelbild                                     'Variablen ohne Kennung sind Integer
LOCATE 3, 27                                       'Anfang der SUB
PRINT "DAS MEGARECHENPROGRAMM!!!!"               'Titelschrift
LOCATE 5, 21                                       '
PRINT "... aus The Real Adok's Way to QBASIC."    '

CIRCLE (320, 200), 100, 9                           'Kopf
PAINT (320, 200), 9, 9                             '

CIRCLE (280, 166), 30, 14                           'Linkes Auge
PAINT (280, 166), 14, 14                           '

CIRCLE (360, 166), 30, 14                           'Rechtes Auge
PAINT (360, 166), 14, 14                           '

CIRCLE (280, 166), 10, 2                             'Linke Pupille
PAINT (280, 166), 2, 2                             '

CIRCLE (360, 166), 10, 2                             'Rechte Pupille
PAINT (360, 166), 2, 2                             '

LINE (320, 180)-(320, 240), 15                       'Nase

LINE (260, 220)-(320, 260), 4                       'Mund
LINE (320, 260)-(380, 220), 4                       '
END SUB                                             'Ende der SUB

```



- *SUB cPrint*: Als Parameter muß eine Zeichenkette angegeben werden. Diese wird dann zentriert auf dem Bildschirm ausgegeben.
- *FUNCTION Formatted\$*: Als Parameter muß eine Integervariable angegeben werden. Diese wird dann in einen String umgewandelt. Überflüssige Leerzeichen werden abgeschnitten. Diese *FUNCTION* ist besonders gut für die Ausgabe von Zahlen geeignet. Sie läßt sich auch leicht an Fließkommazahlen anpassen.
- *FUNCTION nteWurzel#*: Als Parameter müssen zwei Variablen vom Typ *DOUBLE* angegeben werden. Diese *FUNCTION* haben wir ja schon in der letzten Folge besprochen.
- *FUNCTION ZufallsZahl%*: Als Parameter müssen zwei Integervariablen übergeben werden. Diese bestimmen die Unter- und die Obergrenze für die zu erzeugende Zufallszahl. Die Zufallszahl ist vom Typ *INTEGER*. Der Typ läßt sich aber leicht ändern. Falls Fließkommazahlen erzeugt werden sollen, muß noch zusätzlich der *INT*-Befehl verändert werden.

## Nachwort

Das war auch schon *The Real Adok's Way to QBASIC*. Ich hoffe, daß es euch Spaß gemacht hat und ihr auch etwas dabei gelernt habt.

Eine kleine Statistik: In 12 Folgen veröffentlichte ich insgesamt 43 Beispielprogramme, gab euch aber nur 8 Hausaufgaben. Unter den Beispielprogrammen befanden sich viele nützliche Programme, darunter auch drei größere Projekte: ein Adventure, ein Rechenprogramm und eine Toolbox.

Beim Erstellen dieses Kurses richtete ich mich nach keinem Buch. Alle Ideen entstanden in meinem Kopf, und alle Beispielprogramme wurden auch nur von mir programmiert. Dieser Kurs ist das Ergebnis meiner langen Programmiererfahrung. Ich hoffe, die Mühe und die Zeit, die ich in den Kurs investiert habe, haben sich gelohnt und euch geholfen, sich mit der Programmiersprache QBasic vertraut zu machen.

Ich bin gerne bereit, euch bei Problemen zu helfen. Mit den Befehlen und Funktionen, die ihr in diesem Kurs erlernt habt, könnt ihr schon eine ganze Menge anfangen. Alles andere hängt von eurer Vorstellungskraft, Phantasie und den Ideen ab.

Mit dem Erlernen einer Programmiersprache eröffnet sich euch eine neue Welt. Nun kann man mit dem Computer kommunizieren, ihm seine eigenen Ideen verständlich machen und seine ungeahnte Möglichkeiten für eigene Zwecke nützen.

Jeder, ob in der Schule oder im Beruf, ist laufend mit verschiedenen Problemen konfrontiert, die er mit Hilfe des Computers wesentlich vereinfachen und lösen kann. Man soll aber nicht dabei bleiben, nur die Routinearbeiten zu erledigen. Wichtig wäre es, neue Ideen umzusetzen, etwas Neues zu schaffen.

Wenn man eine Programmiersprache erlernt hat, heißt es aber noch lange nicht, daß man nichts mehr Neues dazulernen kann. Es gibt hier keine Grenzen. Alles hängt von euch ab. Versucht, neue Nischen zu finden, riskiert etwas, setzt euch ein, gebt euch mit dem Erreichten nicht zufrieden!

Vielleicht sehen wir einander wieder in einem anderen Kurs. Ich habe noch so viele Ideen...