Facharbeit in Informatik

QuickBASIC – Tutorial

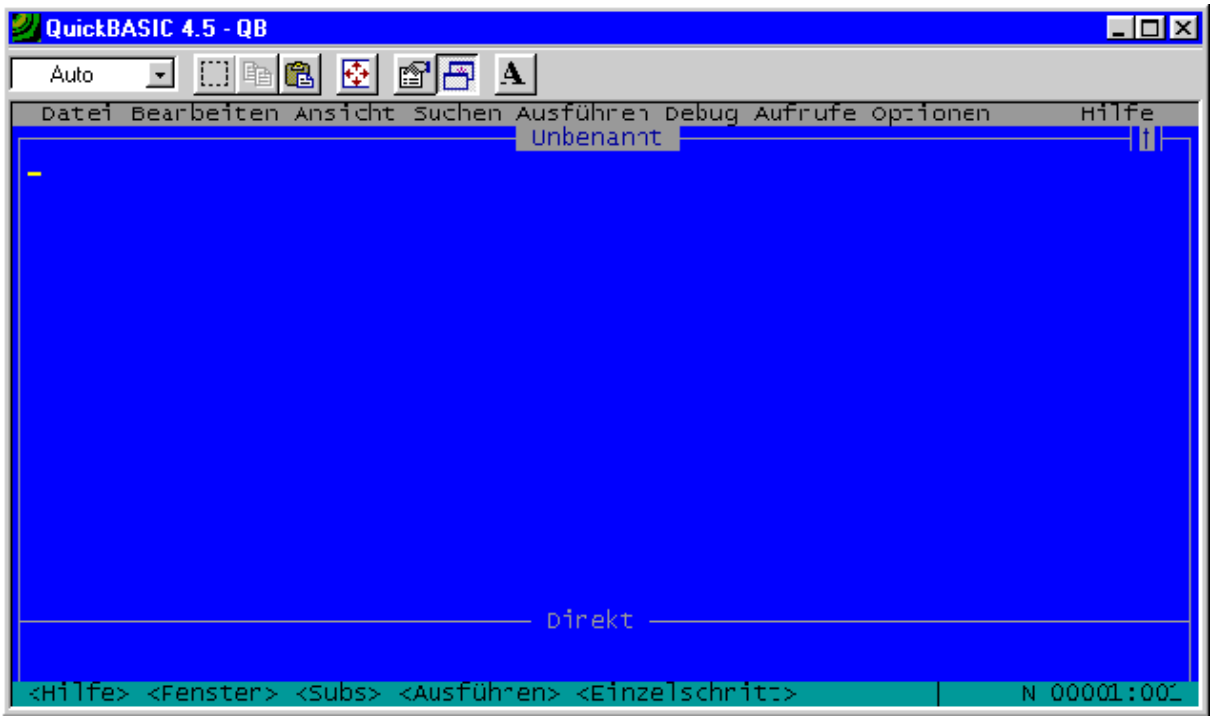
verfasst von Marty Winkler, 11/b
mailto:martywinkler@gmx.net

PDF-Konvertierung von Thomas Antoni (mailto:thomas@antonis.de
http://www.qbasic.de +++ Hottest Qbasic Stuff on Earth

Inhaltsverzeichnis

1 Einleitung

Die Programmiersprache BASIC (Abkürzung für: Beginners All-purpose Symbolic Instruction Code, zu deutsch: "Universelle Programmiersprache für Anfänger") wurde 1962 von JOHN G. KEMENY und THOMAS E. KURTZ auf dem Darthmouth-College in New Hampshire (USA) entwickelt. Seitdem entstanden unzählige Dialekte, wie zum Beispiel Amiga-, GW-, BASIC-A, oder eben Q- und QuickBASIC. BASIC ist so aufgebaut, dass mit wenigen, leicht erlernbaren Befehlen auch umfangreiche Programme erstellt werden können. In den nachfolgenden Kapiteln möchte ich Ihnen die Bedienung des BASIC-Interpreters und die wichtigsten Befehle erläutern.



Screenshot des QuickBASIC 4.5-Interpreters

2 Grundlegendes

2.1 Bedienung des QuickBASIC-Interpreters

In dieser Facharbeit beziehe ich mich hauptsächlich auf die Version 4.5 des Microsoft BASIC-Interpreters. Zwar laufen die meisten Beispielprogramme auch mit der Version 1.1, aber QuickBASIC 4.5 hält eine umfangreichere Hilfe und deutlich mehr Beispiele zu den verschiedenen Befehlen bereit. Ein weiterer Vorteil dieser Version liegt darin, dass mit ihr der Quellcode (das BASIC-Programm) in ein ausführbares Programm (Dateiendung EXE) umgewandelt werden kann, welches ohne den BASIC-Interpreter ausgeführt werden kann und somit spürbar schneller läuft. Die deutsche Version von QuickBASIC 4.5 können Sie im Internet auf der Seite http://www.antonis.de/ herunterladen.

Starten Sie den Interpreter indem Sie die Datei QB.EXE ausführen. Normalerweise lächelt Ihnen jetzt ein blauer Bildschirm entgegen. Wenn Sie in der Menüzeile auf 'Optionen' gehen öffnet sich ein Menüfenster. Wählen Sie nun 'Bildschirm'. Jetzt können Sie die Textfarbe und die Hintergrundfarbe nach Ihren Wünschen verändern. Um die Einstellungen zu speichern klicken Sie auf 'OK', wenn sie unverändert bleiben sollen, dann klicken Sie auf 'Abbrechen'. Stellen Sie als nächstes sicher, dass das vollständige Menü angezeigt wird ('Optionen/Vollständiges Menü') und die Syntaxüberprüfung ('Optionen/Syntaxüberprüfung') aktiviert ist. Wählen Sie danach den Menüpunkt 'Suchpfade festlegen' und tragen Sie die entsprechenden Pfade ein.

Um ein BASIC-Programm unter einem bestimmten Namen zu speichern wählen Sie Datei/Speichern unter'. Wenn Sie ein BASIC-Programm abspeichern möchten und den Namen beibehalten wollen, so wählen Sie 'Datei/Speichern'. Um das aktuelle Programm auszuführen, drücken Sie die Tasten Umschalt+F5 oder wählen Sie im Menü 'Ausführen' den Punkt 'Start'. So, das waren die wichtigsten Menüpunkte. Wenn Sie genaueres über ihre Bedeutung erfahren möchten, so wählen Sie den entsprechenden Punkt mit den Cursortasten aus und drücken F1.

Der QuickBASIC-Interpreter kann auch als Texteditor "missbraucht" werden. Klicken Sie hierzu auf 'Datei/Datei laden' oder 'Datei/Datei erstellen'. Beachten Sie aber, dass Sie als Dateityp 'Dokument' wählen

2.2 Strukturierung von BASIC-Programmen

Generell beginnt ein BASIC-Programm mit einem Programmkopf. Ein Beispiel zur Verdeutlichung:

```
REM------------------------------------------------------------
REM  Titel: KOPF.BAS
REM  Autor: Marty Winkler
REM  Datum: 22.09.2001
REM------------------------------------------------------------
REM  Aufgabe: Dieses Programm zeigt, wie ein
REM           Programmkopf aussehen kann.
REM------------------------------------------------------------
```

Hier lernen Sie schon den ersten Befehl kennen. Der Befehl *REM* ermöglicht das Einfügen von Kommentarzeilen. Anstelle von *REM* kann auch der Apostroph (') benutzt werden. Kommentarzeilen tragen sehr zur Übersichtlichkeit und zum leichteren Verständnis des Programms bei, daher empfehle ich Ihnen diese Möglichkeit zu nutzten. Um dieses kleine Programm zu starten drücken Sie SHIFT+F5. Und wie Sie sehen . . . sehen Sie nichts.

Das liegt daran, dass unser Programm nur aus Kommentarzeilen besteht. Lediglich in der unteren Zeile des Bildschirms steht die Aufforderung eine Taste zu drücken. Machen Sie dies, so erscheinen die Programmzeilen wieder auf dem Monitor. Um sicherzugehen, dass der Programmtext, Quellcode genannt, auch für andere Personen leicht zu verstehen ist ist es wichtig Befehlsgruppen oder einzelne Befehle einzurücken und Absätze in Form von Leerzeilen zu setzen. Hier gleich noch ein Beispiel:

```
REM------------------------------------------------------------
REM  Titel: FORMAT.BAS
REM  Autor: Marty Winkler
REM  Datum: 22.09.2001
REM------------------------------------------------------------

DO
  INPUT "Passwort eingeben: ", Wort$
LOOP UNTIL Wort$ = "QuickBASIC"
```

Wichtig: Beachten Sie bei der Eingabe des Passwortes die Groß- und Kleinschreibung! Aber keine Angst, wenn Sie das Passwort falsch eingeben werden Sie noch mal aufgefordert es einzugeben. Um das Programm abzubrechen drücken Sie einfach Strg+Untbr.

3 Variablen und Datentypen

3.1 Informationen zu den Datentypen

In QuickBASIC gibt es fünf verschiedene Datentypen. Diese unterscheiden sich erstens im Speicherplatzbedarf und zweitens in der Größe der Zahlen die in ihnen gespeichert werden können.

INTEGER (Ganze Zahlen)	Kennzeichen: %
	Bereich: -32768 bis +32767
	Länge: 2 Byte
LONG (Lange Ganzzahlen)	Kennzeichen: &
	Bereich: -2147483648 bis +2147483647
	Länge: 4 Byte
SINGLE (Reelle Zahlen)	Kennzeichen: !
	Bereich: ± 1.401298 bis ±3.402823 E+38
	einfache Genauigkeit (7 Ziffern)
	Länge: 4 Byte
DOUBLE (Reelle Zahlen)	Kennzeichen: #
	Bereich: ±4.940656 D-324 bis ±1.7976931 D+308
	doppelte Genauigkeit (15 Ziffern)
	Länge: 8 Byte
STRING (Zeichenketten)	Kennzeichen: $
	Bereich: 0 bis 32767 Zeichen
	Länge: Anzahl der Zeichen

3.2 Variablen

Das folgende Kapitel befasst sich mit Datentypen und Variablen. Variablen beziehen sich auf eine bestimmte Zahl, eine Zeichenkette oder ein Datenfeld. Der Name einer Variablen darf maximal 40 Zeichen lang sein und folgende Zeichen enthalten:

A-Z, a-z, 0-9

Wichtig ist, dass der Variablenname **immer** mit einem Buchstaben beginnen muss! Und nun einige Beispiele:

```
HausNummer% = 3                 'die Variable 'HausNummer' ist vom
                                'Typ INTEGER und hat den Wert 3

Einwohner% = 160000             'FALSCH, weil 'Einwohner' vom Typ
                                'INTEGER ist, der maximale Wert
                                'für INTEGER-Variablen ist aber
                                '32767

Einwohner& = 160000             'so ist es richtig

Stadt01$ = "Seelow"             ' 'Stadt01' ist vom Typ STRING und
                                'enthält den Text, der in den
                                'Anführungszeichen steht, also:
                                'Seelow

Code$ = "5D2a"                  'Zeichenketten, Strings genannt,
                                'können auch Zahlen und
Zeichen$ = "() ?) # * !“        'andere Zeichen enthalten

Preis! = 2.34                   'Typ: SINGLE, hier sind auch
                                'Kommastellen möglich
```

Wenn die meisten Variablen in einem Programm vom Typ INTEGER sind, können Sie sich die Kennzeichnung dieser Variablen sparen, indem sie als erste Programmzeile folgenden Befehl schreiben:

```
DEFINT A-Z
```

Das heißt, dass alle Variablen die nicht anders gekennzeichnet wurden als INTEGER-Variablen gelten. Der zweite Teil des Befehls (A-Z) bezeichnet die Variablen die als INTEGER definiert werden sollen. In diesem Fall sind es diejenigen, deren Namen mit A, B, C, D ... oder Z beginnen.

```
DEFLNG A, B-D, F, X-Z
```

Alle Variablen deren Name mit den Buchstaben A, B, C, D, F, X, Y oder Z beginnen gelten als Variablen des Typs LONG. Hier sind alle möglichen Befehle aufgelistet:

DEFINT	INTEGER
DEFLNG	LONG
DEFSNG	SINGLE
DEFDBL	DOUBLE
DEFSTR	STRING

Wenn QuickBASIC bei der Ausführung eines Programms auf eine Variable stößt, und sie nicht besonders gekennzeichnet wurde, so nimmt QuickBASIC, abhängig von der Größe der Zahl an, dass diese Variable vom Typ LONG, SINGLE oder sogar DOUBLE sei. Dementsprechend werden mindestens 4 Byte für die jeweilige Variable reserviert. Wird die Variable aber so definiert wie sie benötigt wird, dann kann das Speicherplatz sparen. Im Gegensatz zu kleinen Programmen ist das besonders bei großen Programmen wichtig. Ein Programmbeispiel:

```
'mehrere Definitionen sind möglich!
DEFDBL L-P                    'L-P  => DOUBLE
DEFSTR A, H                   'A, H => Zeichenketten (Strings)

Adresse = "Bertolt-Brecht-Straße"
HausNummer% = 3               '% muss sein, weil alle Variablen
                              'mit Anfangsbuchstaben A und H als
                              'STRING definiert wurden
HausNummer = "3"              'oder man schreibt es so
```

Ein anderer Aspekt ist der, dass Variablen vom Typ INTEGER schneller verarbeitet werden können als Variablen des Typs SINGLE oder DOUBLE. Das folgende Programm verdeutlicht diese Geschwindigkeitsunterschiede äußerst anschaulich:

```
DEFINT A-Z

Anfang! = TIMER
  FOR Zaehler# = 0 TO 1000000# STEP 1
     TYPint% = 0
  NEXT
Ende! = TIMER
PRINT "INTEGER  Benötigte Zeit: "; Ende! - Anfang!

Anfang! = TIMER
  FOR Zaehler# = 0 TO 1000000# STEP 1
     TYPsingle! = 0
  NEXT
Ende! = TIMER
PRINT "SINGLE   Benötigte Zeit: "; Ende! - Anfang!

Anfang! = TIMER
  FOR Zaehler# = 0 TO 1000000# STEP 1
     TYPdouble# = 0
  NEXT
Ende! = TIMER

PRINT "DOUBLE   Benötigte Zeit: "; Ende! - Anfang!
```

In diesem Fall wird der Variablen TYPint der Typ INTEGER (durch %), TYPsingle der Typ SINGLE (durch !) und der Variablen TYPdouble der Typ DOUBLE zugewiesen. Nehmen wir den ersten Teil des Programms doch einmal auseinander:

DEFINT A-Z	Alle Variablen sind vom Typ INTEGER.
Anfang! = *TIMER*	Die Variable Anfang ist vom Typ einfache Genauigkeit (SINGLE). Der Befehl *TIMER* gibt die Anzahl der Sekunden nach Mitternacht zurück. In der Variable Anfang werden hier also die Sekunden nach Mitternacht gespeichert.
FOR Zaehler# = 0 *TO* 1000000# *STEP* 1	Diese Befehlszeile leitet eine sogenannte *FOR-TO-NEXT*-Schleife ein. Wenn man diese Zeile ins Deutsche übersetzt, so würde sie etwa folgendermaßen lauten: "Für Zaehler = 0 bis Zaehler = 1000000 tue:". Zaehler wird nach jedem Durchlaufen der Schleife um die angegebene Schrittweite erhöht, in diesem Fall 1. Diese Schrittweite wird durch *STEP* angegeben. *STEP* kann auch weggelassen werden, dann beträgt die Schrittweite automatisch 1.
TYPint% = 0	In dieser Programmzeile wird festgelegt, dass die Variable TYPint vom Datentyp INTEGER zu sein hat. Weiterhin wird der Variablen der Wert 0 zugewiesen.
NEXT	Mit diesem Befehl wird die *FOR-TO-NEXT*-Schleife wieder geschlossen. Das heißt, dass alle Befehle, die zwischen *FOR* und *NEXT* stehen so oft wiederholt werden, bis Zaehler den Wert 1.000.000 enthält. Trifft der QuickBASIC-Interpreter also auf *NEXT*, so erhöht er die Variable Zaehler um die Schrittweite (hier 1). Stellt er danach aber fest, dass Zaehler größer ist als 1000000, so bearbeitet er den nächsten Befehl. In diesem Fall ist das die folgende Zeile.
Ende! = *TIMER*	Ist die Ausführung der Schleife beendet, so wird die aktuelle Anzahl der Sekunden seit Mitternacht in Ende! gespeichert.
PRINT "INTEGER Benötigte Zeit: "; Ende! – Anfang!	Der Befehl *PRINT* wird im nächsten Kapitel ausführlich behandelt, deswegen erwähne ich jetzt nur, dass mit ihm der String der in den Anführungszeichen steht auf dem Bildschirm ausgedruckt wird. Steht dieser Text dann auf dem Monitor, so berechnet QuickBASIC das Ergebnis folgender Rechenaufgabe: Ende! – Anfang! Danach wird auch das auf dem Bildschirm ausgedruckt, und zwar in der gleiche Zeile wie der Text. Das Ergebnis sind hier die Anzahl der Sekunden, die das Programm benötigt hat, um die Schleife 1.000.000 mal zu durchlaufen.

Wenn sie einen sehr schnellen Rechner haben, die errechneten Werte sich also kaum unterscheiden, so sollten sie die Anzahl der Schleifendurchläufe (1.000.000) erhöhen. Beachten Sie dabei aber, dass die Zahl im gültigen Bereich liegt (siehe Kapitel 3.1 Datentypen). Führt man dieses Programm schließlich aus, so sieht man deutlich, dass die Schleife, in der mit einer INTEGER-Variablen gearbeitet wird, schneller verarbeitet werden kann als die mit SINGLE- oder DOUBLE-Variablen. Als ich dieses Programm auf meinem Rechner (Windows 98, 133 MHz) ausführte, bekam ich folgende Werte:

```
INTEGER  Benötigte Zeit: 5
SINGLE   Benötigte Zeit: 8,558594
DOUBLE   Benötigte Zeit: 8,902344
```

Geben Sie einmal folgendes Programm ein und führen Sie es aus:

```
DEFINT A-Z

Zensur1 = 3                    ' 1. Zensur
Zensur2 = 2                    ' 2. Zensur
Zensur3 = 2                    ' 3. Zensur
Anzahl = 3                     ' Anzahl der Zensuren

Durchschnitt = (Zensur1 + Zensur2 + Zensur3) / Anzahl
PRINT Durchschnitt
```

In diesem Programm werden zuerst alle Variablen als INTEGER deklariert. Danach werden vier Variablen Werte zugewiesen. Als nächstes wird der Durchschnitt errechnet, indem alle Zensuren addiert werden und dann durch die Anzahl der Noten dividiert werden. Das Ergebnis wird in der Variable Durchschnitt abgespeichert. Mit dem Befehl *PRINT* wird der Inhalt von Durchschnitt auf dem Bildschirm ausgedruckt. Aber da steht seltsamerweise eine 2, doch (3 + 2 + 2) durch 3 ergibt ungefähr 2.333. Wie kommt also solch ein Ergebnis zustande? Zu Beginn des Programms haben wir festgelegt, das alle Variablen ganze Zahlen (INTEGER) zu sein haben. Was heißen soll, dass auch Durchschnitt eine Ganzzahl ist. Das heißt, dass das Ergebnis bevor es in der Variable Durchschnitt gespeichert werden kann zu einer ganzen Zahl gerundet werden muss. In diesem Fall wurde abgerundet (2.33... zu 2).Versuchen Sie es jetzt mit diesem Programm und schauen Sie, was dieses Mal angezeigt wird:

```
DEFINT A-Z
CLS                            'CLearScreen = Bildschirm löschen
                               '(näheres im Kapitel 4)
Zensur1 = 3                    '1. Zensur
Zensur2 = 2                    '2. Zensur
Zensur3 = 2                    '3. Zensur
Anzahl = 3                     'Anzahl der Zensuren

Durchschnitt1 = (Zensur1 + Zensur2 + Zensur3) / Anzahl
PRINT Durchschnitt1

Durchschnitt2& = (Zensur1 + Zensur2 + Zensur3) / Anzahl
PRINT Durchschnitt2&

Durchschnitt3! = (Zensur1 + Zensur2 + Zensur3) / Anzahl
PRINT Durchschnitt3!

Durchschnitt4# = (Zensur1 + Zensur2 + Zensur3) / Anzahl
PRINT Durchschnitt4#
```

4 Textausgabe und Formatierungsmöglichkeiten

In diesem Kapitel geht es nun endlich ans Programmieren. Als erstes werde ich den *PRINT*-Befehl ("Drucken") behandeln. Dieser Befehl wird benutzt um Werte (unter anderem von Variablen) oder Texte auf dem Bildschirm auszugeben.

```
PRINT "10 + 10 sind "          'einen Text ausgeben
PRINT 10+10                    'Wert ausgeben
```

Die Hochkommas am Ende der beiden Zeilen teilen QuickBASIC mit, dass danach Kommentare folgen (siehe *REM*). Der erste *PRINT*-Befehl gibt den Text aus der in den Anführungszeichen steht. Beim zweiten *PRINT*-Befehl wird erst das Ergebnis (10+10=20) berechnet und dieses dann auf dem Bildschirm ausgegeben. Sie haben aber auch die Möglichkeit das Ergebnis in die gleiche Zeile, in welcher der Text steht, zu schreiben. Setzten Sie einfach ein Semikolon (;) oder ein Komma nach dem letzten Anführungszeichen und starten Sie das Programm mit SHIFT+F5 neu.

Dieses Beispielprogramm zeigt die verschiedenen Kombinierungsmöglichkeiten und deren Wirkungen:

```
DEFINT A-Z

PRINT "Darstellung Nr."        'Text ausgeben
PRINT 1                        'Zahl 1 an den Anfang der nächsten
                               'Zeile ausgeben

PRINT "Darstellung Nr.";       'Text ausgeben; das Semikolon teilt
                               'QuickBASIC mit, dass die nächste
                               'Bildschirmausgabe hinter den
                               'gerade geschriebenen Text
                               'geschrieben werden soll

PRINT 2                        'Zahl an der aktuellen
                               'Cursorposition ausgeben (direkt
                               'nach dem gerade geschriebenen
                               'Text)

PRINT "Darstellung Nr."; 2     'genau dasselbe wie oben

PRINT "Darstellung Nr.",       'Das Komma setzt die aktuelle
                               'Cursorposition auf den nächsten
                               'Tab-Stop. Ein Stop befindet sich

PRINT 3                        'bei jeder 14. Bildschirmspalte

PRINT "Darstellung Nr.", 3     'das ist genau dasselbe wie oben

PRINT "Darstellung Nr. 4"      'selbstverständlich kann die Zahl
                               'auch in den Anführungszeichen
                               'stehen!
```

Anstatt der Zahl kann auch eine Variable stehen:

```
DEFINT A-Z

Text$ = "Hausnummer:"          'Variablen mit Werten
                               initialisieren
Nummer = 3

PRINT Text$, Nummer            'verschiedene Möglichkeiten den
                               'PRINT-Befehl mit Variablen zu
                               'verwenden, anstatt

PRINT "Nummer:"; Nummer        'des Semikolons kann auch ein Komma
                               'benutzt

PRINT Text$; "3"               'werden, probieren Sie es aus!!
```

Es können auch mehrere Semikolons und/oder Kommas in einer *PRINT*-Anweisung verwendet werden. Zum Beispiel um mehrere Variablenwerte in einer Zeile anzuzeigen und Übersichtlichkeit im Programmtext zu wahren:

```
DEFINT A-Z

Strasse$ = "Bertolt-Brecht-Straße"
Nummer = 3

PRINT "Straße: "; Strasse$; " Hausnummer: "; Nummer
' oder so:
PRINT "Straße: "; Strasse$; Nummer
```

Der nächste Befehl mit dem ich Sie vertraut machen möchte heißt *CLS* (CLear Screen = "Bildschirm reinigen"). Lassen Sie das letzte Beispielprogramm noch einmal laufen, so sehen Sie, dass der Text unter dem schon vorhandenen Text geschrieben wurde. Fügen Sie jetzt in die Zeile über der ersten *PRINT*-Anweisung den Befehl *CLS* ein, und starten Sie das Programm erneut. Nun wird der Text oben in die linke Ecke des Bildschirms geschrieben.

Wie aber kann man einen beliebigen Text an einer bestimmten Position auf dem Bildschirm ausgeben? Zunächst einmal muss erwähnt werden, dass der Textmodus in der Breite 80 Zeichen und in der Höhe 25 Zeichen darstellen kann. Das Zeichen in der linken oberen Ecke befindet sich in der ersten Spalte und der ersten Zeile. Das Zeichen darunter befindet sich in der selben Spalte, aber in der zweiten Zeile. Der Befehl, mit dem man festlegen kann, an welcher Position die nächste Bildschirmausgabe erfolgen soll, lautet *LOCATE* ("platzieren"). Die Befehlsschreibweise können Sie dem folgenden Beispielprogramm entnehmen:

```
CLS                            'Bildschirm löschen (säubern)

LOCATE 10, 38                  'Position: 10. Zeile, 38. Spalte

PRINT "Zeile 10, Spalte 38"    'Text schreiben

PRINT "Zeile 11, Spalte 1"     'dieser Text erscheint in der
                               'nächsten Zeile am linken
                               'Bildschirmrand
```

5 Farbgestaltung im Textmodus

5.1 Vordergrundfarbe

Bisher sahen die Beispielprogramme langweilig, weil farblos, aus. In diesem Kapitel möchte ich versuchen dies zu ändern. Versuchen Sie sich doch gleich einmal an folgendem Programm:

```
DEFINT A-Z

SCREEN 0
CLS

FOR Farbe = 0 TO 15
   COLOR Farbe
   PRINT "Farbnummer:"; Farbe
NEXT

COLOR 7
```

DEFINT A-Z	Alle Variablen sind vom Typ INTEGER
SCREEN 0	Mit dem Befehl *SCREEN 0* wird der Textmodus aktiviert.
CLS	Bildschirm reinigen
FOR Farbe = 0 *TO* 15	"Für Farbe = 0 bis Farbe = 15 tue:"
COLOR Farbe	In dieser Befehlszeile wird die Schriftfarbe festgelegt. Eine Auflistung der Farben finden Sie unter dieser Tabelle.
PRINT "Farbnummer:"; Farbe	Anzeigen der Farbnummer
NEXT	Hier wird die Schleife geschlossen
COLOR 7	Standardfarbe wiederherstellen (Schriftfarbe auf hellgrau setzen)

Und hier sind wie versprochen die Farbnummern für den Textmodus:

0	Schwarz	4	Rot	8	Dunkelgrau	12	Hellrot
1	Blau	5	Magenta	9	Hellblau	13	Hellviolett
2	Grün	6	Braun	10	Hellgrün	14	Gelb
3	Zyan	7	Hellgrau	11	Hellzyan	15	Weiß

Es gibt auch die Möglichkeit, den Text zum Blinken zu bringen. Und zwar addiert man einfach 16 zu der entsprechenden Farbnummer hinzu. Die blinkenden Versionen der jeweiligen Farben erhalten Sie , wenn Sie die folgende Befehlszeile

```
FOR Farbe = 0 TO 15
```

durch diese ersetzen:

```
FOR Farbe = 16 TO 31
```

5.2 Hintergrundfarbe

Natürlich kann man auch die Hintergrundfarbe des Textes festlegen. Standardmäßig ist immer die Farbe Schwarz eingestellt (Farbnummer 0). Um dies zu demonstrieren, verändern wir einfach das letzte Programm um ein paar Kleinigkeiten:

```
DEFINT A-Z

SCREEN 0
CLS

FOR HFarbe = 0 TO 7
   LOCATE 1, 1
   COLOR , HFarbe
   FOR VFarbe = 0 TO 7
      COLOR VFarbe
      PRINT "VFarbe:"; VFarbe; SPC(5); "HFarbe:"; HFarbe
   NEXT
   Anfang! = TIMER
   DO: LOOP UNTIL TIMER > Anfang! + 1
NEXT

COLOR 7, 0
```

Aus den letzten beiden Unterkapiteln geht also folgende Befehlsschreibweise für den *COLOR*-Befehl hervor:

COLOR Vordergrundfarbe, Hintergrundfarbe

Zu Beachten ist allerdings, dass für Hintergrundfarbe nur Zahlen im Bereich von 0 bis 7 zulässig sind. Die Farben, die den Farbnummern zugewiesen sind, können Sie der vorigen Tabelle entnehmen.

Der Befehl *SPC(*5*)* macht nichts anderes als fünf Leerzeichen auf dem Bildschirm auszugeben. Diese kann man natürlich nicht sehen. Dafür sieht man, dass der Text, welcher danach ausgedruckt wird um fünf Stellen nach rechts verschoben wird. Eine weitere Veränderung sind diese zwei Befehlszeilen:

```
   Anfang! = TIMER
   DO: LOOP UNTIL TIMER > Anfang! + 1
```

Die erste dieser beiden Zeilen habe ich schon einmal in Kapitel 3.2 erklärt. Und zwar wird in Anfang die Zahl der Sekunden, die seit Mitternacht vergangen sind gespeichert. Die zweite Zeile stellt ein Konstrukt dar. Die Anweisung *DO-LOOP-UNTIL* ("Wiederhole die Schleife bis ein bestimmtes Ereignis eintritt") macht nichts anderes als die Befehle, welche sich innerhalb von *DO* und *LOOP* befinden, solange zu wiederholen, bis *TIMER* größer als Anfang! plus 1 ist. Auf deutsch heißt das, dass die Schleife solange wiederholt wird bis eine Sekunde vergangen ist. In diesem Fall befinden sich zwischen *DO* und *LOOP* keine Befehle, es wird also nur gewartet.

6 Daten über die Tastatur eingeben

6.1 Die Anweisung INPUT

Das ist ja alles schön und gut, aber wie kann man Zahlen oder Namen in QuickBASIC von der Tastatur entgegennehmen? Dies kann man unter anderem mit der Funktion *INPUT* ("Eingabe") bewerkstelligen. Und hier folgt gleich ein Beispiel:

```
REM-----------------------------------------------------------------
REM  Titel: INPUT01.BAS
REM  Autor: Marty Winkler
REM  Datum: 29.12.2001
REM-----------------------------------------------------------------

DEFINT A-Z

SCREEN 0
CLS

INPUT "Name:"; Name$                        'Beispiel 1
INPUT "Alter:"; Alter                       'Beispiel 2
INPUT "Wohnort, PLZ:"; WohnOrt$, PLZ        'Beispiel 3

PRINT
PRINT "Sie heißen "; Name$; ", sind"; Alter; "Jahre alt und"
PRINT "wohnen in "; WohnOrt$
```

Wie Sie sicher bemerkt haben gibt es mehrere Möglichkeiten die *INPUT*-Funktion zu benutzen. Eines haben alle drei Beispiele gemeinsam: es wird ein Text auf dem Monitor ausgegeben. Und zwar muss dieser Text genau wie bei der *PRINT*-Anweisung in Anführungszeichen stehen. Danach folgt ein Semikolon und die Variable in der die Eingabe gespeichert werden soll, wobei es egal ist von welchem Typ die Variable ist. Bei der Eingabe muss allerdings beachtet werden, dass man keine Buchstaben oder andere Sonderzeichen eingibt, wenn eine Zahl verlangt wird. Was passiert, wenn man es doch macht? Probieren Sie es einfach mal aus, kaputt gehen kann nichts.

Das dritte Beispiel springt ein bisschen aus der Reihe, weil in dieser Befehlszeile nach mehreren Informationen gefragt wird! Man muss also in diesem Fall bei der Eingabe den Wohnort von der Postleitzahl durch ein Komma trennen. Sonst weiß QuickBASIC nicht welcher Teil der Eingabe zu welcher Variablen gehört.

Nun hat QuickBASIC aber nach dem Text Fragezeichen ausgegeben! Das ist dazu da, dass der Anwender weiß er hat eine Eingabe vorzunehmen. Diese Fragezeichen sehen aber eigentlich nicht schön aus. Das wussten wohl auch die Entwickler von QuickBASIC. Was heißen soll, dass man das Fragezeichen dadurch verhindern kann, indem man anstatt eines Semikolons ein Komma benutzt. Ändern Sie das Programm jetzt so um, dass die lästigen Fragezeichen verschwinden. So, nun sieht es doch schon viel angenehmer aus, oder?

Jetzt haben wir aber ein kleines Problem, denn was geschieht, wenn Sie nur ein Jahr alt sind? Dann denkt man als Anwender, dass der Programmierer dieser "Software"

Schwierigkeiten mit der deutschen Sprache hat. Denn in diesem Fall steht auf dem Bildschirm "..., sind 1 Jahre alt ...". Wie kann man dieses Problem beheben?

Dazu muss ich Sie mit einer neuen Anweisung bekannt machen, der *IF-THEN-ELSE*-Anweisung ("wenn ... dann tue... ansonsten tue..."). Schauen Sie sich einfach das nächste Beispiel an:

```
REM-----------------------------------------------------------
REM  Titel: INPUT02.BAS
REM  Autor: Marty Winkler
REM  Datum: 29.12.2001
REM-----------------------------------------------------------

DEFINT A-Z

SCREEN 0
COLOR 7, 4
CLS

COLOR 14, 1
PRINT SPC(30); "Die INPUT-Anweisung"; SPC(31);
PRINT                                        'Leerzeile drucken

COLOR 7, 4
INPUT "Name:", Name$
INPUT "Alter:", Alter
INPUT "Wohnort, PLZ:", WohnOrt$, PLZ

IF Alter = 1 THEN Text$ = "Jahr" ELSE Text$ = "Jahre"

PRINT                                       'Leerzeile drucken
PRINT "Sie heißen "; Name$; ", sind"; Alter; Text$; " alt und"
PRINT "wohnen in "; WohnOrt$
```

Wortwörtlich übersetzt heißt die hier gebrauchte Form folgendes:

> Wenn die Variable Alter den Wert 1 hat, dann wird in der Variablen Text$ die Zeichenkette "Jahr" gespeichert. Ansonsten enthält Text$ den String "Jahre".

Es gibt aber noch eine Schreibweise für die *IF-THEN-ELSE*-Anweisung. Die nachstehende wird meist dann verwendet, wenn mehrere Befehle auf einmal ausgeführt werden sollen und dies dann nicht mehr in eine einzige Befehlszeile passt.

```
IF Alter = 1 THEN
    Text$ = "Jahr"
ELSE
    Text$ = "Jahre"
END IF
```

Wenn man diese Form verwendet, so muss am Ende des *IF*-Blocks *END IF* stehen, um QuickBASIC zu signalisieren, dass die *IF-THEN-ELSE*-Anweisung hier zuende ist.

6.2 Die Funktionen INKEY$ und INPUT$(n)

Möchte man nur ein Zeichen einlesen, oder kontrollieren ob der Benutzer eine Taste drückt, so kommt die Funktion *INKEY$* ins Spiel. Wenn eine Taste gedrückt wird, dann liefert *INKEY$* das entsprechende Zeichen zurück. Wurde aber keine Taste gedrückt, so liefert diese Funktion eine leere Zeichenkette ("") zurück.

```
DEFINT A-Z

RANDOMIZE TIMER                'Zufallsgenerator initialisieren

SCREEN 0                       'Textmodus
COLOR 7, 1: CLS                'Vordergrund: grau
                               'Hintergrund: blau
LOCATE 3, 1
PRINT TAB(7); "Versuchen Sie so schnell wie möglich das Zeichen,"
PRINT TAB(7); "welches in der Bildschirmmitte erscheint
                               einzutippen."

PRINT
PRINT TAB(7); "Aber zunächst müssen Sie eine Schwierigkeitsstufe
                               auswählen:"
PRINT TAB(7); "1] Leicht, 2] Mittel, 3] Schwer"

DO
  Wahl$ = INKEY$                                'ASCII-Zeichen:
  IF Wahl$ = "1" THEN Min = 97: Max = 122      ' von a bis z
  IF Wahl$ = "2" THEN Min = 65: Max = 125      ' von A bis }
  IF Wahl$ = "3" THEN Min = 33: Max = 125      ' von ! bis }
LOOP UNTIL Max <> 0

Anfang1! = TIMER

   FOR Zaehler = 1 TO 20       '20 Zeichen eintippen
      Zeichen = INT((Max - Min + 1) * RND + Min)
                               'einzutippendes Zeichen per
                               'Zufallsgenerator auswählen
      LOCATE 10, 40: COLOR 14: PRINT CHR$(Zeichen)
                               'auf dem Bildschirm ausgeben
      Anfang2! = TIMER
      DO
        LOCATE 1, 60
        COLOR 2                'Vordergrundfarbe: grün
        PRINT USING "###.##"; TIMER - Anfang2!
                               'alle Ziffern hinter den ersten
                               'beiden Kommastellen nicht anzeigen
      LOOP UNTIL INKEY$ = CHR$(Zeichen)
   NEXT

Ende1! = TIMER
Durchschnitt! = (Ende1! - Anfang1!) / 20
                               'Durchschnitt errechnen

COLOR 12                       'Vordergrundfarbe: hellrot
LOCATE 10, 40: PRINT " "       'letztes Zeichen löschen

PRINT TAB(7); "Pro Zeichen benötigten Sie durchschnittlich "
PRINT TAB(7); USING "###.##"; Durchschnitt!;
                               'alle Ziffern hinter den ersten
PRINT TAB(7); "Sekunden!";     'beiden Kommastellen nicht anzeigen
```

Hier folgt eine kurze Erklärung:

RANDOMIZE TIMER	Initialisiert den Zufallsgenerator mit der Anzahl der Sekunden seit Mitternacht. Ohne diesen Befehl würde es bei jeder Ausführung des Programms die gleiche Abfolge von Zeichen geben.
TAB(n)	Versetzt den Cursor in die Spalte n.
RND	Liefert eine Zahl zwischen 0 und 1. Um eine Zahl in einem bestimmten Bereich (mit der Obergrenze Max und der Untergrenze Min) zu erhalten muss die Formel welche hier benutzt wurde verwendet werden.
CHR$(n)	Gibt das ASCII-Zeichen mit der Nummer n zurück. Informationen dazu finden Sie in jedem Tafelwerk oder Sie bewegen den Cursor auf den Befehl *CHR$()* und drücken die Taste F1. Danach müssen Sie doppelt auf **ASCII-Zeichen-Codes** klicken.
PRINT USING "###.##"; Variable	Von Variable werden alle Ziffern, die vor dem Komma stehen, angezeigt. Aber nur die ersten beiden Kommastellen.

Die *INPUT$(n)*-Funktion ("Eingabe") wartet auf die Eingabe von genau n Zeichen. Wenn man weniger Zeichen eingeben möchte muss man nach der Eingabe die Enter-Taste drücken. Im Gegensatz zu der Funktion *INPUT* werden keine Zeichen auf dem Bildschirm angezeigt, genauso verhält es sich auch bei der *INKEY$*-Funktion. Man könnte also anstatt

```
DO:LOOP WHILE INKEY$ = ""
```

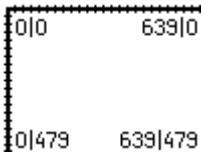
auch folgendes schreiben:

```
Variable$ = INPUT$(1)
```

7 Farbgestaltung im Grafikmodus

Jetzt möchte ich Sie mit den wichtigsten Grafikbefehlen bekannt machen. In den Beispielprogrammen kommt nur der Grafikmodus mit der Nummer 12 zum Einsatz.

Er hat eine Auflösung von 640 Bildschirmpunkten (Pixel) in Richtung der X-Achse und 480 Bildschirmpunkten in Richtung der Y-Achse. Dabei liegt der Koordinatenursprung (X=0, Y=0) in der oberen linken Ecke des Monitors.



Aber bevor wir die Grafikbefehle nutzen können, müssen wir natürlich in den gewünschten Modus umschalten. Dies geschieht mit dem selben Befehl den wir auch für den Textmodus verwendet haben.

Der einfachste Grafikbefehl ist *PSET* (Point SET, "Punkt setzen"). Wie es die Übersetzung schon verlauten lässt verleiht er einem bestimmten Pixel eine gewünschte Farbe. Die Koordinaten folgen direkt auf das Befehlswort *PSET*, wobei zuerst der x-Wert und danach der y-Wert anzugeben ist. Auf die Koordinaten folgt die Farbnummer die Sie der Tabelle aus dem Kapitel 5.1 entnehmen können. Zusätzlich gibt es die Möglichkeit mit *STEP* relative Koordinaten anzugeben. Aber schauen Sie sich einfach folgendes Beispiel an:

```
  SCREEN 12
  CLS

'--- Überschrift
  COLOR 11
  PRINT TAB(31); "Die PSET-Anweisung"
  PRINT

'--- absolute Koordinaten (320, 240)
  COLOR 7
   PRINT "Einen weißen Punkt mit den Koordinaten (320, 240)
                                setzen:"
  COLOR 9
   PRINT TAB(10); "PSET (320, 240), 15"

   PSET (320, 240), 15

 Warten$ = INPUT$(1)
'--- relative Koordinaten (320, 240) + (10, 10) = (330, 250)
'                         letzte Ko. + rel. Ko. = aktuelle Ko.
  COLOR 7
   PRINT "Relativ zu den letzten Koordinaten einen gelben Punkt
                                setzen:"
  COLOR 9
   PRINT TAB(10); "PSET STEP(10, 10), 14"

   PSET STEP(10, 10), 14

 Warten$ = INPUT$(1)
'--- wieder relative Koor. (330, 250) + (-10, -10) = (320, 240)
```

```
'                          letzte Ko. + rel. Ko.   = aktuelle Ko.
  COLOR 7
   PRINT "Den ersten Punkt löschen (= Farbe schwarz):"
  COLOR 9
   PRINT TAB(10); "PSET STEP(-10, -10),0"

   PSET STEP(-10, -10), 0

 Warten$ = INPUT$(1)
  COLOR 7
```

Der nächste Befehl soll die *LINE*-Anweisung ("Linie") sein. Diese zeichnet Linien oder Rechtecke auf den Bildschirm. Auch dazu ein Beispiel:

```
  SCREEN 12
  CLS

'--- Überschrift
  COLOR 11
  PRINT TAB(31); "Die LINE-Anweisung"
  PRINT

'--- Linie
  COLOR 7
   PRINT "eine weiße Linie von (500, 300) bis (600, 400):"
  COLOR 9
   PRINT TAB(10); "LINE (500, 300)-(600, 400), 15"

   LINE (500, 300)-(600, 400), 15

 Warten$ = INPUT$(1)
'--- relative Koordinaten
  COLOR 7
   PRINT "relative Koordinatenangabe:"
  COLOR 9
   PRINT TAB(10); "LINE STEP(0, -100)-(500, 400), 15"

   LINE STEP(0, -100)-(500, 400), 15

 Warten$ = INPUT$(1)
'--- ein Rechteck: B = border ("Rahmen")
  COLOR 7
   PRINT "ein gelbes Rechteck:"
  COLOR 9
   PRINT TAB(10); "LINE (500, 300)-(600, 400), 14"

   LINE (500, 300)-(600, 400), 14, B

 Warten$ = INPUT$(1)
'--- ein ausgefülltes Rechteck: BF = border filled ("gefüllter
                              Rahmen")
  COLOR 7
   PRINT "ein gefülltes rotes Rechteck:"
  COLOR 9
   PRINT TAB(10); "LINE (480, 270)-(620, 299), 4"

   LINE (480, 270)-(620, 299), 4, BF

 Warten$ = INPUT$(1)
'--- ein andere Möglichkeit eine Linie zu zeichnen
  COLOR 7
   PRINT "ein andere Möglichkeit eine Linie zu zeichnen:"
```

```
  COLOR 9
   PRINT TAB(10); "LINE -(620, 430), 4"

   LINE -(620, 430), 4

 Warten$ = INPUT$(1)
'--- und zuende zeichnen
  COLOR 7
   PRINT "und zuende zeichnen:"
  COLOR 9
   PRINT TAB(10); "LINE -(480, 430), 4"
   PRINT TAB(10); "LINE -(480, 270), 4"

   LINE -(480, 430), 4
   LINE -(480, 270), 4

 Warten$ = INPUT$(1)
 COLOR 7
```

Um Kreise zu zeichnen benutzt man den Befehl *CIRCLE* ("Kreis"). Auch hier sind relative Koordinatenangaben möglich.

```
  SCREEN 12
  CLS

'--- Überschrift
  COLOR 11
  PRINT TAB(31); "Die CIRCLE-Anweisung"
  PRINT

'--- Linie
  COLOR 7
   PRINT "einen grauen Kreis bei (500, 300) mit einem Radius von
                              30:"
  COLOR 9
   PRINT TAB(10); "CIRCLE (500, 300), 30, 7"

   CIRCLE (500, 300), 30, 7

 Warten$ = INPUT$(1)
'--- relative Koordinaten
  COLOR 7
   PRINT "einen gelben Kreis bei (460, 300) mit einem Radius von
                             10:"
  COLOR 9
   PRINT TAB(10); "CIRCLE STEP(-40, 0), 10, 14"

   CIRCLE STEP(-40, 0), 10, 14

 Warten$ = INPUT$(1)
 COLOR 7
```

Der vorletzte Grafikbefehl den ich Ihnen vorstellen möchte heißt *PAINT* ("färben"). Mit ihm kann man einen bestimmten Bereich mit der angegebenen Farbe ausfüllen . . . also färben.

```
  SCREEN 12
  CLS

'--- Überschrift
  COLOR 11
```

```
  PRINT TAB(31); "Die PAINT-Anweisung"
  PRINT

'--- Rechteck
  COLOR 7
   PRINT "zuerst ein gelbes Rechteck zeichnen (50*50 Pixel):"
  COLOR 9
   PRINT TAB(10); "LINE (300, 350)-STEP(50, 50), 14, B"

   LINE (300, 350)-STEP(50, 50), 14, B

 Warten$ = INPUT$(1)
'--- und jetzt ausfüllen
  COLOR 7
   PRINT "und jetzt mit Blau (1) ausfüllen, bis QuickBASIC auf
                              Gelb (14) stößt:"
  COLOR 9
   PRINT TAB(10); "PAINT (325, 375), 1, 14"

   PAINT (325, 375), 1, 14

 Warten$ = INPUT$(1)
 COLOR 7
```

Um die Sache abzurunden kann man nicht nur einem Punkt eine Farbe verleihen, sondern auch die Farbe eines bestimmten Punktes ermitteln. Dies geschieht mit der Funktion *POINT* ("Punkt"):

```
DEFINT A-Z
RANDOMIZE TIMER

  SCREEN 12
  CLS

'--- Überschrift
  COLOR 11
  PRINT TAB(31); "Die POINT-Funktion"
  PRINT

'--- Rechteck
  COLOR 7
   PRINT "zuerst einen Punkt mit einer zufälligen Farbe
                              zeichnen:"
  COLOR 9
   PRINT TAB(10); "PSET (300, 300), INT(16 * RND)"

   PSET (300, 300), INT(16 * RND)

 Warten$ = INPUT$(1)
'--- und jetzt Farbe ermitteln
  COLOR 7
   PRINT "und jetzt die Farbe ermitteln:"
  COLOR 9
   PRINT TAB(10); "Farbe = POINT(300, 300)"
   PRINT TAB(10); "PRINT Farbe"

   Farbe = POINT(300, 300)
   PRINT Farbe

 Warten$ = INPUT$(1)
 COLOR 7
```

Anhang

A Linkliste

Hier folgen Links der Seiten, auf die ich beim Surfen im Internet getroffen bin, und die ich Ihnen nicht vorenthalten werde.

www.Gymnasium-Seelow.de	Die Homepage des Gymnasiums, welches ich besuche. Hier können Sie QuickBASIC-Programme herunterladen, die ich programmiert habe.
www.antonis.de	Eine Q-BASIC-Seite mit wahnsinnig vielen Programme zum downloaden
www.qb-fun.de	Eine sehr übersichtliche Seite mit Links zu anderen Q-BASIC-Seiten und Programmen sowie nützlichen Bibliotheken zum downloaden.

Zum Schluss meine E-Mail-Adressen, falls Sie Fragen oder Probleme haben, mich auf Fehler hinweisen oder Verbesserungsvorschläge machen möchten. Ebenfalls willkommen sind mir selbstprogrammierte QuickBASIC-Programme.

KuhBasic@gmx.net

oder

martywinkler@gmx.net

B Literaturverzeichnis

QuickBASIC 4.5 Online-Hilfe, downloadbar bei http://www.antonis.de/

Buch und Zeit Verlagsgesellschaft, Computer Lexikon von A bis Z, Seite 42, o. O., 1995

C Erklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benützt habe.