Qbasic Kurs

Gymnasium Wolbeck

Zusammenfassungen der Unterrichtsstunden

Autor: Daniel Hampf (http://www.hampfi.de)
PDF-Umsetzung: Thomas Antoni (http://www.qbasic.de)

Inhalt

I. Stunde (5.Sep. 2001)

1. Variablen

- Variablen speichern in einem Programm eine Zahl oder einen Text (eine sog. "Zeichenkette", engl.: "*String*")
- Eine Variable hat in QBasic einen Namen, der mit einem Buchstaben beginnen muss und keine Sonderzeichen enthalten darf.
- Das letzte Zeichen gibt die Art der Variable an:
    - String - Variablen haben als Endung ein Dollarzeichen $
    - Zahlenvariablen haben unterschiedliche Endungen, je nachdem wie gross die Zahlen werden dürfen und ob Kommastellen gespeichert werden sollen. Für den Anfang reichen sog. *Integers* (Ganzzahlvariablen, bis 32500, ohne Kommastellen). Sie haben als Endung das Prozentzeichen %
- Variablen bekommen einen Wert durch ein Gleichheitszeichen zugewiesen:

```
zahl1% = 5
antwort$ = "Addieren"
```

2. Befehle

Befehle sind bestimmte Wörter, die in QBasic beim Programmablauf eine bestimmte Aktion auslösen, wenn sie aufgerufen werden. Jeder Befehl sollte in eine neue Zeile geschrieben werden.

Fast alle Befehle haben sog. *Parameter*, die bestimmen, was genau der Befehl machen soll. Beispiel: Der Befehl PRINT schreibt einen Text auf den Bildschirm. Welchen Text er schreibt, muss man als Parameter übergeben, etwa PRINT "Hallo".

Ob und wenn welche Parameter übergeben werden müssen, steht in der sog. *Syntax*.

3. Der PRINT-Befehl

Syntax: `PRINT text$`
oder: `PRINT zahl%`

Beschreibung: Schreibt einen Text auf den Bildschirm und stellt den Schreibcursor in die nächste Zeile (d.h. der Text vom nächsten PRINT - Befehl wird in die nächsten Zeile geschrieben)

Besonderheiten: Will man mehrere Variablen hintereinander ausgeben, werden die durch Semikolon getrennt. Trennt man durch normale Kommas, lässt das Programm zwischen den Variablen auf dem Bildschirm eine Spalte frei (ähnlich wie beim Tabulator in Windows)

4. Der CLS-Befehl:

Syntax: `CLS`

Beschreibung: Löscht den Inhalt des Bildschirms, also etwa alles was vorher durch PRINT - Anweisungen ausgegeben worden ist.

5. Variablen mit Formeln zuweisen

Variablen kann man nicht nur einen Wert zuweisen, sondern auch eine Formel, die das Programm dann ausrechnet, etwa:

```
summand1% = 78
summand2% = 9
summe% = summand1% + summand2%
```

In Summe% wäre jetzt der Wert 87 gespeichert.

6. Input Befehl

Syntax: `INPUT Aufforderung$, antwort$`
oder: `INPUT Aufforderung$, zahl%`

Beschreibung: Fordert vom Benutzer eine Eingabe.

Beispiel:

```
aufforderung$ = "Geben Sie Ihren Namen ein: "
INPUT aufforderung$, name$
PRINT "Ihr Name ist "; name$
```

Die Variable name$ hat in diesem Fall die Eingabe des Benutzers gespeichert, und man kann diesen Inhalt jetzt abrufen und (z.B.) ausgeben.

7. Der IF ... THEN - Befehl

Syntax: `IF zahl% = 4 THEN PRINT text$`
oder:
```
IF zahl% = 4 `THEN
  CLS
  PRINT "Hallo"
  PRINT "du"
END IF
```

Beschreibung: Führt Befehle aus, wenn eine Bedingung *wahr* ist. In diesem Fall wird etwa der Text nur ausgedruckt wenn zahl% wirklich den Wert 4 enthält. Hat man mehrere Befehle, empfielt es sich, diese untereinander zu schreiben und den Block mit END IF zu beenden.

II. Stunde (12. Sep. 2001)

8. Der SELECT CASE Befehl

Syntax:
```
SELECT CASE var%
  CASE 1: <Befehle>
  CASE 2: <Befehle>
  CASE ELSE: <Befehle>
END SELECT
```

Beschreibung: Mit dieser Struktur kann eine Variable, die mehrere verschiedene Werte annehmen kann, leichter und übersichtlicher abgefragt werden als vielen aneinandergereihten IF .. THEN ... Befehlen. Zusätzlich erleichert der CASE ELSE Befehl die Behandlung von unvorhergesehenden Ereignissen.

9. Der END Befehl

Syntax: `END`

Beschreibung: Normalerweise hört ein Programm auf, wenn es keine weiteren Befehle mehr findet. Will man zwischendrin schon mal aufhören, kann man gut den END Befehl einsetzen, der das Programm sofort und ohne weitere Umschweife beendet.

10. Der GOTO Befehl

Syntax:
```
GOTO sprungmarke
...
...
...
sprungmarke:
```

Beschreibung: Veranlasst das Programm, an eine andere Stelle zu springen, die sowohl weiter oben als auch weiter unten sein kann. Die Sprungmarke kann natürlich irgendwie heissen, sollte aber zweckmäßigerweise ungefähr angeben, wofür sie da ist (genau wie Variablen), also etwa *wiederholung* oder *ende* oder so. Baut man die Sprungmarke vor den GOTO Befehl, dann hat man eine Schleife, bei der man möglichst auch eine Möglichkeit zum Beenden der Schleife bauen sollte. Insgesamt machen GOTO Befehle eigentlich nur Sinn, wenn sie irgendwie bedingt sind, das heisst nur ausgeführt werden, wenn eine bestimmte Bedingung erfüllt ist.

11. Zufallszahlen

Für viele Bereiche, ganz besonders aber für Spiele jeglicher Art, braucht man immer Zufallszahlen. Damit einem grundsätzlich erstmal Zufallszahlen zur Verfügung stehen, muss man irgendwo am Anfang des Programmes den Befehl `RANDOMIZE TIMER` benutzen. Dieser Befehl macht noch nix, er ist nur zur sog. Initialisierung (quasi Bereitstellung).
Will man dann eine Zufallszahl benutzen, muss man eine Variable haben, in der diese Zufallszahl gespeichert ist, nennen wir sie zufall%:

```
zufall% = RND * 6
```

Jetzt ist in der Variable eine Zahl zwischen 0 und 5 gespeichert. Der Befehl kann beliebig abgewandelt werden, um einen größeren oder kleineren Zahlenbereich abzudecken.

III. Stunde (26. Sep. 2001)

12. Funktionen

Funktionen sind Befehle, die einen Wert zurückgeben, oder man könnte auch sagen, die ein Ergebnis liefern. Dieses Ergebnis muss natürlich gespeichert werden, damit die Befehle Sinn machen. Das sieht

dann etwa so aus: ergebnis% = BEFEHL
Beispiele sind diese Befehle:

```
zz% = RND * 5                    'Bestimmt eine Zufallszahl
ganzzahl% = INT (3.1416)         'Rundet eine Zahl, bzw. schneidet
Nachkommastellen ab
text$ = LCASE$ (erstertext$)     'Wandelt einen Text in
Kleinbuchstaben um
```

13. Grafikbefehle

`SCREEN modus%`
Initialisiert den Bildschirm, das heisst bereitet die Grafikausgabe vor. Muss vor allen anderen Grafikbefehlen stehen, am besten am Anfang vom Programm. modus% bestimmt die Auflösung, gute Werte sind 9 oder 12.

`PSET (x%, y%), f%`
Zeichnet an die Stelle x%, y% auf den Bildschirm einen Punkt (Pixel).
x% geht von links (0) bis rechts (640); y% geht von oben (0) bis unten (480).
f% gibt die Farbe an, gültig sind i.d.R. 0 bis 15.

`CIRCLE (x%, y%), r%, f%`
Zeichnet einen Kreis in der Farbe f% an die Koordinaten x%, y%.
r% ist der Radius in Pixeln.

`LINE (x1%, y1%) - [STEP] (x2%, y2%), f% [,BF]`
Zeichnet eine Linie.
Erstmal gilt: Alles was in eckigen Klammern steht, kann man weglassen (so was heißt dann *optional*)
x1% und y1% sind die Koordinaten vom Anfangspunkt der Linie, x2% und y2% vom Endpunkt; f% ist wieder die Farbe.
Benutzt man STEP, so sind x2% und y2% *relativ*, d.h. sie werden zu den Anfangswerten dazugerechnet. Beispiel:

```
LINE (100, 100) - STEP (50, 50), 4
```

zeichnet eine rote Linie von 100, 100 bis 150, 150.

Benutzt man B oder BF, so entsteht anstatt einer Linie ein Rechteck, bzw. ein ausgefülltes Rechteck.

`PAINT (x%, y%), f%, randfarbe%`
Malt den Bildschirm oder einen Teil desselbigen aus. Fängt an dem Punkt x%, y% an und mal so lange, bis er auf die Farbe randfarbe% stösst. Fängt man also in einem Kreis oder Viereck an zu zeichnen und hat als Randfarbe die Farbe des Kreises bzw. Vierecks angegeben, so malt er nur die Figur aus. Hat die Figur eine Öffnung, oder hat man als Randfarbe einen falschen Wert , so wird sofort der ganze Bildschirm ausgemalt. Will man übrigens den ganzen Bildschirm ausmalen, so braucht man keine Randfarbe angeben (dürfte klar sein oder ?).

Alle Grafiken können übrigens mit CLS wieder gelöscht werden. Oder dadurch, das man sie genau nochmal malt, und dann in der Hintergrundfarbe,, z.B. schwarz. (Zweite Methode ist zwar vielleicht umständlich, aber deutlich schneller).

IV. Stunde (24. Oktober 2001)

14. Relative Koordinaten

```
LINE (x1%, y1%) - STEP (x2%, y2%)
```

zeichnet eine Linie, die am Punkt (x1%; y1%) beginnt. Der Endpunkt der Linie ist der Punkt (x1% + x2% ; y1% + y2%). Die Koordinaten des Zielpunkts sind also *relativ*.
QBasic merkt sich immer, wo der letzte gemalte Punkt ist. Hat man z.B. die Zeile

```
LINE (100, 80) - STEP (10, 15)
```

zeichnet es eine Linie von 100, 80 bis 110, 95.
Diese letzten Koordinaten gelten dann als Ausgangskoordinaten für weitere STEP-Anweisungen, z.B.

```
CIRCLE STEP (50, 50), 10
```

zeichnet einen Kreis (vom Radius 10) um den Mittelpunkt 160, 145.

ebenso eine weitere LINE Anweisung:

```
LINE STEP(0, 0) - STEP (100,100)
```

Dieser Befehl malt eine Linie vom Mittelpunkt des Kreises 160, 145 zu 260, 245.

15. Unterprogramme

Zur besseren Übersicht benutzt man oft Unterprogramme. Um ein neues Unterprogramm zu erstellen wählt man aus dem Menü Edit --> New Sub und gibt einen Namen für das Unterprgramm ein. Es erscheint ein Bildschirm, der etwa so aussieht:

```
SUB malen ()

END SUB
```

Zwischen diese beiden Befehle kann man nun alle normalen QBasic Befehle schreiben, die dann ausgeführt werden, wenn man das Unterprogramm aufruft. Um ein Unterprogramm aufzurufen, schreibt amn den Namen einfach ins Hauptprogramm (oder in ein anderes Unterprogramm).

16. Parameter

Variablen aus dem Hauptprogramm sind in einem Unterprogramm grundsätzlich nicht bekannt. Will man dem Unterprogramm bestimmte Werte mitteilen, so kann man sie hinter den Aufruf schreiben:

```
malen 100, 120
```

Im Unterprgramm werden diese Werte dann in die Variablen geschrieben, die in Klammern hinter dem Prozedurnamen stehen:

```
SUB malen (xpos%, ypos%)
...
END SUB
```

Diese Variablen sind dann im Unterprogramm verfügbar, aber auch nur dort, im Hauptprogramm sind die dann wieder weg.

17. Globale Variablen

Will man bestimmte Variablen in allen Unterprogrammen und im Hauptprogramm benutzen, dann kann man sie als *global* definieren. Dazu schreibt man an den Anfang des Programms:

```
DIM SHARED var%
```

Die Variable var% ist dann global bekannt.

V. Stunde (31.Oktober 2001)

18. Dateien bearbeiten

Um eine Datei zu lesen, muss diese zunächst geöffnet werden:

```
OPEN "example.txt" FOR INPUT AS #1
```

Jetzt kann mit verschiedenen Befehlen aus der Datei gelesen werden. Ich empfehle diesen:

```
INPUT #1, var$
```

Dabei wird ein Teil der Datei in die Variable var$ gelesen (Wenn man eine Zahl erwartet solte da übrigens auch besser var% stehen).
Wenn man alles aus einer Datei gelesen hat, muss man die Datei wieder schließen (ansonsten kann die Datei beschädigt werden):

```
CLOSE
```

Damit hat man schon alles, was man zum Dateilesen braucht. Die Befehle zum Schreiben einer Datei sehen ähnlich aus:

```
OPEN "example.txt" for OUTPUT AS #1
WRITE #1, var$
CLOSE
```

Dabei wird der Text aus der Variable var$ in die Datei geschrieben.

19. Tastatureingaben abfragen

Wenn der Benutzer eine Taste drückt, wird diese Taste in INKEY$ gespeichert. Den Wert von INKEY$ kann man am besten mit einer SELECT CASE Strukur abfragen:

```
SELECT CASE INKEY$
  CASE "q": END
  CASE "a": PRINT "Du hast die Taste a gedrückt"
  CASE ELSE: PRINT "Du hast eine andere Taste gedrückt"
END SELECT
```

So eine Strukur muss fast in jedem Programm oder Spiel vorhanden sein, das Tastatureingaben abfragt. Damit die Eingaben nicht nur einmal, sondern ständig abgefragt werden, muss diese Strukur in einer DO-LOOP-Schleife stehen.

20. Strings

Strings sind Variablen, die Text enthalten, also alle Variablen mit der Endung $. Hier einige Befehle zur Behandlung von Strings:

```
anz = LEN (text$)          'Speichert die Anzahl der Buchstaben (=die
                           ' Länge) von text$ in die Variable anz
wert = ASC (zeichen$)      'Speichert die ASCII-Zahl eines
Buchstabens
zeichen$ = CHR$ (zahl%)    'Speichert den Buchstaben, der zur
angegebenen ASCII-Zahl gehört
buchstabe$ = MID$ (text$, anfang, länge) 'Speichert einen Ausschnitt
                                         ' aus einem text$
```

VI. Stunde (7.Nov. 2001)

21. Variablentypen

Es gibt noch einige Variablentypen mehr außer den euch bisher bekannten:

- String (Text) mit der Endung "$"
- Integer - Zahl (Ganzzahl, ohne Komma) mit der Endung "%"

Weitere:

- Zahl einfacher Genauigkeit (Kommazahl bis 10^38) mit der Endung "!" . Dieser Variablentyp wird auch automatisch erzeugt wenn man keine Endung an die Variable dranhängt.
- Zahl doppelter Genauigkeit (lange Kommazahl bis 10^308) mit der Endung „#„

22. Schleifen

Es gibt theoretisch vier Möglichkeiten, ein Schleife zu basteln:

1) Man setzt eine Sprungmarke und ruft diese dann mit `GOTO` auf
2) Man nimmt die Befehle `WHILE ... WEND`

Diese beiden Möglichkeiten sind aber nicht zu empfehlen: Die erste ist zu langsam und zu unübersichtlich, die zweite ist veraltet und wird über kurz oder lang nicht mehr eingesetzt werden (und auch nicht mehr in die neuen Basic-Versionen eingebaut)

3) Eine DO ... LOOP Schleife. Dieser Typ wird i.d.R. genommen, wenn man noch nicht genau weiss, wie oft die Schleife durchlaufen wird.
4) Eine FOR-NEXT-Schleife. Dieser Typ wird in der Regel genommen, wenn man schon weiss oder berechnen kann, wie oft die Schleife durchlaufen wird. Die Konstruktion sieht so aus:

```
FOR n% = 1 TO 10
  <Befehle>
NEXT
```

Diese Schleife würde 10 mal durchlaufen, dazu hat man den Wert von n%, der angibt, im wievielten Durchlauf man sich befindet.

Hier noch mal eine Möglichkeit für das Verschlüsselungsprogramm von letzter Stunde (mit FOR-NEXT-Schleife):

```
 1  CLS
 2  INPUT "Text: ", text$
 3
 4  FOR n = 1 TO LEN(text$)
 5   st$ = MID$(text$, n, 1)
 6
 7   wert = ASC(st$)
 8   wert = wert + 10
 9
10   code$ = code$ + CHR$(wert)
11  NEXT
12
13  PRINT code$
```

Hinweis: Dieses Programm liest einen Text ein, arbeitet Zeichen für Zeichen ab und wandelt jedes Zeichen in das Zeichen ein, dass in der ASCII-Tabelle zehn Zeichen weiter hinten steht.

23. Wartezeiten

Wenn man Wartezeiten in sein Programm einbauen will, hat man mehrere Möglichkeiten. Hier sind drei:

1) Der Befehl SLEEP x bewirkt dass das Programm für x Sekunden angehalten wird. Vorteil (oder vielleicht manchmal auch Nachteil): Man kann die Wartezeit mit einem Tastendruck abbrechen.

2) Man baut eine kleine Schleife, die keine Befehle enthält, sondern nur sehr oft durchläuft, etwa so:

```
FOR w = 0 TO 1000
NEXT
```

Je nach Rechnergeschwindigkeit dauert es einen Moment bis diese Schleife abgearbeitet ist und das Programm die nächsten Befehle in Angriff nimmt.

3) Wartezeiten bis hinunter zu 0.056 s erzeugt man am elegantesten mit der TIMER-Funktion. Diese stellt den Inhalt des PC-Systemtimers zur Verfügung, der alle 0.056 s erhöht und um Mitternacht auf Null zurückgesetzt wird. TIMER liefert eine einfach lange Gleitpunktzahl (SINGLE) zurück und hat einen Wertebereich von 0.000 bis 86400.000s. Eine Wartezeit! lässt sich wie folgt erzeugen:

```
Endzeit! = TIMER + Wartezeit!
DO
LOOP WHILE TIMER < Endzeit!
```

Diese Programmpassage demonstriert außer der Zeitenbildung auch, wie man eine `DO...LOOP`-Schleife verwendet.

VII. Stunde

In der 7. Stunde wurde nichts Weltbewegendes behandelt, sondern hauptsächlich nur der Stoff der vorhergehenden Stunden wiederholt und vertieft.

VIII. Stunde (21.11.2001)

24. Datenfelder

Datenfelder sind eins der schwierigen Kapitel der Programmierung, aber ganz sicher auch mit Abstand das Allerwichtigste. Datenfelder dienen kurz gesagt dazu, in einer Variable mehrere Werte zu speichern. Als Beispiel nehmen wir jetzt mal die Namen einer Fußballmannschaft. Sicher könnte man jeden Namen unter einer anderen Variablen speichern, aber es geht wesentlich einfacher mit Datenfeldern.

```
DIM spielername$(11)

spielername$(1) = "Frank Rost"
spielername$(2) = "Frank Verlaat"
spielername$(3) = "Frank Baumann"
spielername$(4) = "Krstajic"
spielername$(5) = "Dieter Eilts"
spielername$(6) = "Thorsten Frings"
spielername$(7) = "Viktor Spripnik"
spielername$(8) = "Ronny Ernst"
spielername$(9) = "Lisztes"
spielername$(10) = "Ailton"
spielername$(11) = "Marco Bode"
```

Bis dahin ist durch die Datenfelder zugegebenermaßen nicht viel gewonnen. Der Vorteil liegt jetzt aber in der einfacheren Verwendung dieser Namen. Wenn man z.B eine Spielerliste haben will, kommt man mit drei Zeilen aus:

```
FOR n% = 1 TO 11
  PRINT spielername$(n%)
NEXT
```

Oder wenn man jetzt zufällig einen Spieler auswählen will (z.B. einen Torschützen):

```
zz% = INT(RND * 10) + 2
PRINT spielername$(zz%)
```

Aufgabe:

Lass es schneien !

- Dazu sollen einzelne Pixel (oder kleine Kreise) auf den Bildschirm angezeigt werden, dessen Koordinaten in Datenfeldern gespeichert sind.
- Die Anfangskoordinaten werden zufällig bestimmt.
- Bei jedem Schleifendurchlauf werden die y-Werte aller Koordinaten etwas erhöht, die alten Schneeflocken schwarz übermalt und die neuen in weiss auf den Bildschirm gemalt.
- Wenn eine Schneeflocke unten aus dem Bildschirm rausfällt, muss sie wieder nach oben gesetzt werden (y=0).
- Lösung im Anhang 2 (letzte Seite)

IX. Stunde (21.11.2001)

25. Bausteine zur Arbeit mit Datenfeldern

Man braucht immer zwei Variablen: Das Datenfeld und eine Variable, die die Anzahl der Elemente im Datenfeld speichert. Beispiel:

```
DIM daten$ (0 TO 20)
anzahl% = 0
```

damit kann man schon eine ganze Reihe Möglichkeiten demonstrieren, die man mit Datenfeldern hat:

1) Element hinzufügen:
```
INPUT "Neues Element: ", new$
daten$ (anzahl%) = new$
anzahl% = anzahl % + 1
```

2) Alle Elemente anzeigen
```
FOR n = 0 TO anzahl% - 1
  PRINT daten$(anzahl%)
NEXT
```

3) Element löschen
```
INPUT "Welches Element löschen ?", del%
daten$ (del%) = ""
FOR n = del% TO anzahl% - 1
  daten$ (n) = daten$ (n + 1)
NEXT
anzahl% = anzahl% - 1
```

4) Element einfügen
```
INPUT "Neues Element: ", new$
INPUT "An welcher Stelle", stelle%
FOR n = anzahl% TO stelle% STEP -1
  daten$(n) = daten$(n - 1)
NEXT
daten$ (stelle%) = new$
anzahl% = anzahl% + 1
```

5) Elemente sortieren

```
DIM ndaten$(anzahl)
anzahl = 4
daten$(0) = "Albert"
daten$(1) = "Cecilia"
daten$(2) = "Bob"
daten$(3) = "Dirk"
'
m = 0
FOR n = 0 TO anzahl - 1
  FOR m = 0 TO n - 1
    IF daten$(n) < ndaten$(m) THEN
      'Einfügen
      FOR i = anzahl - 2 TO m STEP -1
        ndaten$(i + 1) = ndaten$(i)
      NEXT
      EXIT FOR
    END IF
  NEXT
  ndaten$(m) = daten$(n)
NEXT

FOR n = 0 TO anzahl - 1
  daten$(n) = ndaten$(n)
NEXT
```

Anhang 1: QBasic-Quiz

Und zum Abschluss gibt es ein kleines Qbasic-Quiz:

Erkläre die folgenden Begriffe:

- Befehl
- Variable
- Programm
- Algorithmus
- Debugging
- Funktionen (Unterschied zw. Informatik/mathe)
- Subs (subroutines)
- Integer
- String
- Array
- Dimensionen
- Index
- Schleifen (4 Möglichkeiten)
- Parameter
- Sprungmarke
- Kollisionsabfrage
- Engine
- Vektoren
- Screen Modes
- Relative Koordinaten
- Globale / Lokale Variablen

Anhang 2: Schneeflockensimulationsprogramm

```
'*******************************************
' Qbasic - Schneeflockensimulationsprogramm
' Lösung der Übungsaufgabe zur VIII. Stunde
'*******************************************
CLS
SCREEN 12
'
anz = 80
'
DIM x(anz), y(anz)
'
RANDOMIZE TIMER
'
'Schneeflockenverteilung am Anfang
FOR n = 0 TO anz
  x(n) = INT(RND * 640)
  y(n) = INT(RND * 480)
NEXT
'
DO
  'Untergrund
  LINE (0, 450)-(640, 480), 15, BF
  'Schneeflocken zeichnen
  FOR n = 0 TO anz
    CIRCLE (x(n), y(n)), 1, 0
    y(n) = y(n) + INT(RND * 4) - 1
    x(n) = x(n) + INT(RND * 3) - 1
    IF y(n) >= 450 THEN
      y(n) = 0
      x(n) = INT(RND * 640)
    END IF
    CIRCLE (x(n), y(n)), 1, 15
  NEXT
LOOP UNTIL INKEY$ = CHR$(27)
```