

# GW-BASIC-Tutorial

Von Thomas Köpplmayr — [www.gwbasic.at](http://www.gwbasic.at)

## Inhalt

- Folge 1 - Der Beginn eines langen Kurses
  - Folge 2 - Verweise und Unterprogramme
  - Anhang 1 - Die wichtigsten GW-BASIC-Befehle
  - Anhang 2 - Die GW-BASIC-Entwicklungsumgebung
  - Anhang 3 - Weitere Informationsquellen zu GW-BASIC
- 

## Folge 1 - Der Beginn eines langen Kurses ...

Am Anfang benötigst du einmal das Programm **gwbasic.exe**, das du dir in meiner Downloadrubrik besorgen kannst. Im folgenden wird angenommen, dass sich GWBASIC im Verzeichnis **C:\gwbasic** befindet. Du kannst es aber auch irgendwo anders hinspeichern, musst nur während des Kurses eventuell darauf achten, dass der Pfad nicht immer mit deinem übereinstimmt.

Das Programm ist eigentlich für DOS gedacht, lässt sich aber bekanntlich auch problemlos unter Windows starten, da ja Windows selbst auf MS-DOS aufbaut. Hat man das Programm erst einmal gestartet, lässt es sich nicht so schnell, ohne eine Warnmeldung zu bekommen, wieder schließen. Du kannst es ruhig einmal ausprobieren, denn wenn du noch nichts geschrieben hast, kann ja keine Information verloren gehen. Auch wenn du bereits etwas programmiert hast, es anschließend ordentlich gespeichert und nun das Fenster wie eine normale Windowsanwendung geschlossen hast, macht das nichts. Eine elegantere Lösung, die auch im Vollbildmodus (wo kein gewöhnliches "Window" zu sehen ist) bzw. im reinen DOS-Modus funktioniert, ist die, einfach das Wort **system** einzugeben und zur Bestätigung die ENTER-Taste zu drücken. Durch diesen Befehl wird die DOS-Anwendung beendet und das Fenster kann normal geschlossen werden.

Die erste Anwendung dieses Kurses soll zuerst den Anwender nach seinem Namen fragen und dann eine kurze Nachricht auf dem Bildschirm ausgeben, die den vorher eingegebenen Namen beinhaltet. Um das zu tun, öffne **gwbasic.exe** und tippe den folgenden Quellcode ein. Vergiss dabei aber nie jede Zeile mit ENTER zu bestätigen, da ansonsten die Zeile nicht angenommen wird!

```
10 CLS
20 PRINT"Wie heißt du?"
30 INPUT NAME$
40 PRINT"Der Anwender heißt ";NAME$;"."
50 END
```

Das erste was du wahrscheinlich wissen möchtest, ist ob das geschriebene Programm auch funktioniert. Dazu musst du das Wort **RUN** in eine neue Zeile schreiben und mit ENTER bestätigen. Alternativ dazu kannst du auch einfach die Taste F2 drücken. Daraufhin wird alles was du zuvor geschrieben hast gelöscht (Keine Angst, es geht dabei nichts verloren!) und es erscheint die Frage **"Wie heißt du?"**. In der nächsten Zeile steht ein Fragezeichen, hinter dem du deinen Namen eingeben sollst. Wenn du als Namen beispielsweise **"Thomas"** eingegeben und wiederum mit ENTER bestätigt hast, müsste dein Bildschirm Folgendes darstellen:

```
Wie heißt du?
? Thomas
Der Anwender heißt Thomas.
Ok
```

Dein Programm wurde zum Schluss bereits beendet und du kannst weitere Befehle eingeben:

## LIST

Mit diesem Befehl wird der gesamte Quelltext wieder angezeigt. Dieser Befehl ist nützlich, um den Quelltext noch einmal zu kontrollieren, falls beim Starten des Programms Fehler auftreten, oder um das Programm zu erweitern bzw. zu verbessern. Bei langen Programmen kommt es vor, dass der Quelltext nicht mehr ganz auf den Bildschirm passt und da man im GW-BASIC Interpreter (**gwbasic.exe**) nicht scrollen, d.h.: die Anzeige nach oben oder unten verschieben, kann. Um nur einen Teil des Quelltextes anzeigen zu lassen, tippe beispielsweise **LIST 10** ein, um die Zeile mit der Nummer **10** anzuzeigen. Es kann natürlich auch eine Gruppe aus mehreren Zeilen angezeigt werden, dazu tippe als Beispiel **LIST 10-30** ein. Um alle Zeilen nach und inklusiv der **20.** anzuzeigen, gib **LIST 20-** ein und um alle Zeilen vor und inklusiv der **20.** anzeigen zu lassen, schreib **LIST -20**. Da der Befehl **LIST** oft gebraucht wird, gibt es dafür eine eigene Taste, nämlich **F1**.

## SAVE

Deine erste Arbeit möchtest du sicher einmal speichern. Tippe dazu **SAVE "name"** ein, um deinem Programm den Namen **"name.bas"** zu geben. Alle in GW-BASIC geschriebenen Programme werden im \*.bas-Format abgespeichert. Wenn du nur den Namen eingibst, wird dein Programm automatisch im selben Ordner gespeichert, in dem sich auch **gwbasic.exe** befindet. Willst du dein Programm in einen anderen Ordner speichern, musst du den gesamten Pfad angeben. Zum Beispiel könntest du mit dem Befehl **SAVE "C:\Meine Programme\name"** dein Programm in den Ordner **"Meine Programme"** auf deiner Festplatte mit dem Kurzzeichen **"C"** speichern. Die Kurztaste zum Speichern ist **F4**.

Um einen Backslash \ einzugeben halte die Taste **Alt Gr** gedrückt und betätige die Taste mit dem scharfen **S**. Bei Pfadangaben wird im Gegensatz zu den Angaben von Internetadressen, bei denen ja bekanntlich ein **Slash /** geschrieben wird, immer ein **Backslash \** verwendet.

## LOAD

Da die in GW-BASIC geschriebenen Programme nicht von selbst laufen, muss man sie zuerst mit dem Interpreter **gwbasic.exe** in den Speicher laden und kann sie dann mit dem Befehl **RUN** ablaufen lassen. Die Regeln für Pfadnamen sind dieselben wie beim Befehl zum Speichern. Befindet sich das Programm im selben Ordner wie **gwbasic.exe**, musst du nur **LOAD "name"** eingeben, ist das nicht so, musst du hingegen beispielsweise **LOAD "C:\Meine Programme\name"** eingeben, um die Datei **"name.bas"** zu laden. Die Kurztaste zum Laden lautet **F3**.

Um nun wieder zum eigentlichen Programm zurückzukommen, drücke **F1** und **ENTER**, falls du das noch nicht getan hast. In der Zeile **10** wird mit dem Befehl **CLS** die gesamte Anzeige gelöscht. In der nächsten Zeile wird der Befehl **PRINT** verwendet, um einen Text am Bildschirm ausgeben zu lassen. In Zeile **30** wird der Befehl **INPUT** verwendet. Mit ihm kann der Benutzer des Programms eine Zahl oder ein Wort eingeben. Diese Zahl bzw. dieses Wort wird dann in einer Variable gespeichert. In diesem Fall wird der Name des Benutzers in der Variable **NAME\$** vorübergehend gespeichert. Variablen mit einem Dollarzeichen hinten dran, nennt man in GW-BASIC Stringvariablen, weil sie nur **Strings**, also Zeichenketten, enthalten können.

Der Unterschied zwischen einer Zahl und einer Zeichenkette liegt im Programmieren allgemein darin, dass man mit Zahlen rechnen kann, mit Zeichenketten allerdings nicht. Das bedeutet also, dass **26** sehr wohl eine Zeichenkette sein kann, man mit ihr aber nicht rechnen kann, da man sie dazu in eine Variable speichern muss, die für Zahlen vorgesehen ist.

In der Zeile mit der Nummer 40 wird wieder ein `PRINT`-Befehl angewandt, der allerdings unterbrochen ist. Würde man `NAME$` innerhalb der Anführungszeichen schreiben, würde der Computer nicht den Inhalt der Variabel, sondern stattdessen einfach das Wort `NAME$` ausgeben. Die Strichpunkte dienen lediglich dazu, um zu verhindern dass GW-BASIC eine neue Zeile beginnt, also den gesamten Satz in ein und derselben Zeile anzeigt. Die Zeile mit der Nummer 40 bewirkt nun, dass zuerst der Satzteil "Der Anwender heißt ", dann in der gleichen Zeile der Inhalt der Variable `NAME$` und zu guter letzt, abermals in der selben Zeile, "." angezeigt wird. Der Punkt muss natürlich wieder unter Anführungszeichen geschrieben werden, da er ja Teil des Textes ist, der mit Hilfe des Befehls `PRINT` angezeigt werden soll. Die letzte Zeile enthält dann noch den Befehl `END`, der bewirkt, dass das jeweilige gerade laufende Programm (nicht aber `gwbasic.exe`) beendet wird. Das erkennt man an dem `Ok` am Schluss des Programms, wenn man es ablaufen lässt.

Was einem bei diesem Programm als erstes auffällt, ist die Tatsache dass jede Zeile eine Nummer hat. Welchen Sinn das Ganze hat, hast du ja schon bei dem Befehl `LIST` erfahren. In den in weiterer Hinsicht folgenden Kursen wirst du noch weitere Anwendungsmöglichkeiten von dieser Zeilennummerierung kennen lernen, doch jetzt ist erstmals nur wichtig, dass Befehle, die nicht Bestandteil des eigentlichen Programms sind (`LIST`, `SAVE`, `LOAD`, `RUN`), keine Zeilennummern erhalten. Im Klartext bedeutet das, dass man jeder Zeile die gespeichert werden soll, eine Zahl zuordnet. Der Befehl `CLS` beispielsweise kann aber auch ohne Zeilennummerierung eingegeben werden, um die Anzeige zu bereinigen. In diesem Fall gehört er aber nicht unmittelbar zum Programm und wird deshalb nicht mitgespeichert, wenn du dein Programm sicherst. Die Abstände zwischen den Zahlen sind egal, sie können auch unregelmäßig sein, müssen allerdings die korrekte Reihenfolge haben. Das heißt, man kann statt 10, 20, 30 und 40 genauso 11, 12, 45 und 115 eingeben. Man darf aber nicht der Zeile mit der Frage nach dem Namen eine höhere Zahl zuordnen als der Zeile mit der Eingabemöglichkeit. Was in Bezug auf die Reihenfolge schon erlaubt ist, wird im folgenden Absatz dargestellt.

```
20 PRINT"Wie heißt du?"
40 PRINT"Der Anwender heißt ";NAME$;". "
50 END
10 CLS
30 INPUT NAME$
```

GW-BASIC ordnet nämlich die Zeilennummern automatisch nach den jeweiligen Zahlen und nicht danach in welcher Reihenfolge die einzelnen Zeilen eingegeben wurden. Wichtig dabei ist nur, dass jede Zeile mit ENTER bestätigt wird. Dieser Zusammenhang wird durch erneutes Auflisten mit dem Befehl `LIST` sichtbar.

Zum Schluss möchte ich dir noch den Befehl `DELETE` zum Löschen einer Zeile vorstellen. Um ihn auszutesten, füge den folgenden Quelltext deinem Programm hinzu:

```
60 PRINT"Bitte lösche mich wieder!"
```

Vergiss nicht ENTER zu drücken und ruf den gesamten Quellcode mit `LIST` auf. Anschließend tippe `DELETE 60` ein und liste abermals den Quelltext auf. Wenn du alles richtig gemacht hast, müsste die gerade hinzugefügte Zeile jetzt wieder verschwunden sein.

## Folge 2 - Verweise und Unterprogramme ...

In diesem Kapitel geht es vor allem darum, wie man in GW-BASIC Verweise auf andere Zeilen oder Unterprogramme in derselben Datei angibt. Dies geschieht durch die beiden Befehle `GOTO [Zeilennummer]` und `GOSUB [Zeilennummer] ... RETURN`, wobei ersterer einen einfachen Sprung in eine andere Zeile und letzterer einen Verweis auf ein Unterprogramm in derselben Datei darstellt. Diese Befehle spielen vor allem in größeren Programmen eine entscheidende Rolle, in denen es darum geht, mehrmals an verschiedenen Stellen im Programm umherzuspringen. Ein Beispiel für ein solches Programm ist ein so genanntes **Text-Adventure**. In einem derartigen Spiel geht es darum, auf Fragen in bestimmten Situationen vorgefertigte Antworten zu geben und so den Spielverlauf zu beeinflussen.

Gib am besten zuerst folgendes Programm ein und speichere es unter einem beliebigen Namen. Optional kannst du es auch aus meiner Downloadrubrik zusammen mit der \*.rtf-Version dieses Kapitels beziehen, um dir die Tipparbeit zu ersparen.

```
10 CLS
20 PRINT"DU BEFINDEST DICH IN EINER KLEINEN ";
30 PRINT"LICHTUNG EINES WALDES."
40 PRINT"VOR DIR STEHT EINE ALTE HUETTE."
50 PRINT"WAS WILLST DU TUN?"
60 PRINT"1 ... HINEINGEHEN"
70 PRINT"2 ... WEITERGEHEN"
80 INPUT"GIB 1 ODER 2 EIN!",X%
90 IF X%=2 THEN GOTO 220
100 IF X%=1 THEN GOSUB 150 ELSE GOTO 80
110 PRINT"RICHTIG!"
120 PRINT"DU HAST NUN DEINE NEUGIERDE BEFRIEDIGT UND"
130 PRINT"KANNST DEINE WANDERUNG BERUHIGT FORTSETZEN."
140 GOTO 240
150 REM UNTERPROGRAMM
160 PRINT"IN DER HUETTE STEHEN EINIGE STUEHLE."
170 INPUT"WIEVIELE SIEHST DU?",Y% 'KLEINES RATESPIEL
180 IF Y%=4 THEN RETURN
190 IF Y%<4 THEN PRINT"ZU WENIG"
200 IF Y%>4 THEN PRINT"ZU VIEL"
210 GOTO 170
220 PRINT"DU GEHST AN DER HUETTE VORBEI"
230 PRINT"UND SETZT DEINE WANDERUNG FORT."
240 PRINT"VIEL SPASS NOCH!"
250 END
```

Um zu sehen, wie das geschriebene Programm arbeitet, lass es einfach einmal ablaufen und folge den Anweisungen auf dem Bildschirm. Zu Beginn bekommst du eine kurze Schilderung der Lage und auch gleich eine entscheidende Frage.

```
DU BEFINDEST DICH IN EINER KLEINEN LICHTUNG EINES WALDES.
VOR DIR STEHT EINE ALTE HUETTE.
WAS WILLST DU TUN?
1 ... HINEINGEHEN
2 ... WEITERGEHEN
GIB 1 ODER 2 EIN!
```

Um mit der kürzesten Möglichkeit anzufangen, nehmen wir einmal an, du tippst **3** ein. GW-BASIC wird dann zur Frage zurückspringen.

```
GIB 1 ODER 2 EIN!3
GIB 1 ODER 2 EIN!
```

Wenn du dir den Code ansiehst, wirst du bemerken, dass die eingegebene Zahl in einem neuen Variablentyp gespeichert wird, einer Ganzzahl zwischen **-32768** und **+32767**. Dieser Typ wird in der Fachsprache auch **Integer** genannt. Gibst du hingegen eine Dezimalzahl oder eine Kombination aus Buchstaben ein, wirst du folgende Fehlermeldung bekommen und abermals aufgefordert werden **1** oder **2** einzugeben:

```
GIB 1 ODER 2 EIN!nix
?Redo from start
GIB 1 ODER 2 EIN!
```

Um jetzt wieder zum vernünftigen Teil zurückzukommen, nehmen wir an, dass du lesen kannst und tatsächlich entweder **1** oder **2** eingibst. Die schnellere Variante in diesem Fall wäre, eine **2** einzugeben und somit die Hütte nicht zu betreten.

```
GIB 1 ODER 2 EIN!2
DU GEHST AN DER HUETTE VORBEI UND
SETZT DEINE WANDERUNG FORT.
VIEL SPASS NOCH!
Ok
```

Somit wurde das Programm beendet und du kommst durch Eingabe von **system** wieder zum System zurück. Dir wird bestimmt schon aufgefallen sein, dass ich den Umlaut **Ü** immer mit **UE** umschrieben habe. Das muss man auf Grund der Tatsache, dass GW-BASIC keine Umlaute und kein scharfes **S** kennt, tun, um zu gewährleisten, dass das jeweilige Wort korrekt angezeigt wird.

Wenn du hingegen eine **1** eingibst, da du die geheimnisvolle Hütte zuerst einmal besichtigen möchtest, gerätst du in ein Unterprogramm. Der Computer sagt dir bloß, dass sich in der Hütte einige Stühle befinden und du sollst die Anzahl der Stühle erraten.

```
IN DER HUETTE STEHEN EINIGE STUEHLE.
WIEVIELE SIEHST DU?
```

Nach einigem Herumprobieren, wirst du bald die richtige Anzahl, nämlich **4**, herausgefunden haben. Doch zuerst noch zu der Möglichkeit, dass du eine falsche Anzahl eingibst:

```
IN DER HUETTE STEHEN EINIGE STUEHLE.
WIEVIELE SIEHST DU?2
ZU WENIG
WIEVIELE SIEHST DU?5
ZU VIEL
WIEVIELE SIEHST DU?
```

Offensichtlich wird dann entweder die Meldung **ZU WENIG** oder **ZU VIEL** ausgegeben und die Frage erscheint erneut. Erst bei Eingabe der Zahl **4** erscheint etwas Neues:

```
WIEVIELE SIEHST DU?4
RICHTIG!
DU HAST NUN DEINE NEUGIERDE BEFRIEDIGT UND
```

```
KANNST DEINE WANDERUNG BERUHGIG FORTSETZEN.  
VIEL SPASS NOCH!  
Ok
```

Das Programm wurde beendet und das letzte Geheimnis gelüftet. Nun zu den neuen Befehlen:

```
90  IF X%=2 THEN GOTO 220  
100 IF X%=1 THEN GOSUB 150 ELSE GOTO 80
```

Der Befehl `IF ... THEN ... ELSE` stellt den einfachsten Fall einer so genannten **Schleife** dar. Die Zeile 90 sagt im Prinzip folgendes aus: Wenn die Variable `X%` (das ist im Übrigen die vorher genannte ganzzahlige Variable) den Wert 2 enthält, dann soll das Programm zur Zeile 220 springen und dort weiterlaufen. Enthält `X%` nicht den Wert 2, so wird die Zeile 100 ausgeführt, die wiederum prüft, ob `X%` den Wert 1 enthält. Ist dies wieder nicht der Fall, tritt der Befehl `GOTO 80` in Kraft.

Ein `IF`-Befehl muss also nicht zwingend ein `ELSE` enthalten, in diesem Fall ist es aber sinnvoll, da die Möglichkeit besteht, dass der Benutzer eine Ganzzahl zwischen -32768 und +32767 eingibt, die weder 1 noch 2 ist. In diesem Fall springt das Programm mit dem Befehl `GOTO 80` einfach zurück zur Fragestellung.

Nun noch zum Unterprogramm: Wie bereits erwähnt, ruft der Befehl `GOSUB` ein Unterprogramm in derselben Datei auf, eine so genannte Sub-Routine. Dieses Unterprogramm wird erst wieder beendet, wenn der Befehl `RETURN` auftaucht. Das alles scheint am Anfang eine unnötige Verkomplizierung zu sein, da man theoretisch ja auch nur `GOTO`-Befehle verwenden könnte, in größeren Programmen wird dies allerdings mit der Zeit sehr unübersichtlich. Es ist hingegen empfehlenswert, Unterprogramme in Gruppen zusammenzufassen und mit `GOSUB`-Befehlen aufzurufen. Große Programme, die nur `GOTO`-Befehle enthalten, nennt man deshalb oft **Spaghetti-Code**, weil durch die dauernde Hin- und Herspringerei ohne zu erkennende Struktur, ein regelrechtes Labyrinth zu Stande kommt.

Ein weiterer Vorteil des `GOSUB`-Befehls besteht darin, dass er mehrere `RETURN`-Befehle enthalten kann, da auf Grund von bestimmten Bedingungen, die beispielsweise durch `IF`-Befehle überprüft werden, an verschiedenen Stellen eines Unterprogramms auftauchen können.

Natürlich kann man auch mehrere `GOSUB`-Befehle verschachteln, ein plötzlich auftauchender `RETURN`-Befehl springt dann einfach zum letzten `GOSUB`-Befehl zurück. Von einer derartigen Verschachtelung ist allerdings abzuraten, da dies natürlich wieder auf Kosten der Übersichtlichkeit geht.

Zum Schluss noch zu dem letzten, unbekannten Befehl:

```
150 REM UNTERPROGRAMM
```

`REM` steht für Remark, was übersetzt Kommentar oder Notiz bedeutet. Mit Hilfe dieses Befehles kann man mitten in einem Programm Kommentare einfügen, die beim Ablauf des Programms nicht beachtet werden. Sie dienen ausschließlich der Hilfe des Programmierers, der vielleicht nach langer Zeit wieder einmal ein komplexeres Programm untersucht und dann nicht mehr genau weiß, wozu er einen bestimmten Befehl eingefügt hat. Es ist also auf alle Fälle empfehlenswert, größere Programme mit Kommentaren auszustatten.

```
170 INPUT"WIEVIELE SIEHST DU?",Y% 'KLEINES RATESPIEL
```



Auch die Zeile 170 enthält ein Kommentar, das nur diesmal nicht durch einen REM-Befehl, sondern durch einen simplen Apostroph eingefügt wurde. Auf diese Weise kann man Kommentare nachträglich auf eine platzsparende Weise einem Programm hinzufügen.

In dieser Zeile wurde außerdem ein INPUT-Befehl mit einem PRINT-Befehl verknüpft. Der Beistrich hat fast dieselbe Funktion wie der Strichpunkt in der letzten Folge. Er verhindert einen Zeilenumbruch in der Darstellung am Bildschirm, mit dem einzigen Unterschied, dass der Beistrich zusätzlich einen Abstand erzeugt und in diesem Fall ein überflüssiges Fragezeichen auf Grund des INPUT-Befehls unterbindet.

Ich habe in diesem Kapitel absichtlich darauf verzichtet, die einzelnen Zeilen des Codes genau zu besprechen und mich mehr auf den Ablauf und die verschiedenen Möglichkeiten eines Ablaufs konzentriert. Ein anderes Vorgehen würde diesen Kurs unnötig in die Länge ziehen und höchstens zu Desinteresse unter den Lesern führen. Auf Basis der genauen Besprechung des Programmablaufs und der Erklärung der Funktion neuer Befehle, dürfte es kein Problem darstellen den Code vollständig zu verstehen. Sollten trotzdem Fragen auftauchen, kontaktier mich einfach über mein Kontaktformular.

Ich wünsche dir noch viel Spaß beim Arbeiten mit GW-BASIC und würde mich freuen, wenn du gespannt auf mein nächstes Kapitel wartest.

## Anhang 1 – Die wichtigsten GW-BASIC-Befehle

- **CLS** - Löscht den Bildschirm
- **END** - Bendet das Programm
- **FOR <Laufvariable> = <Anfangswert> TO <Endwert> ... NEXT <Laufvariable>** - Programmschleife (kann sich über mehrere Zeilen erstrecken)
- **GOSUB <Zeilennummer>** - Sprung in ein Unterprogramm, welches mit RETURN abgeschlossen wird
- **GOTO <Zeilennummer>** - Unbedingter Sprung
- **IF <Bedingung> THEN <Zeilennummer>** - Bedingter Sprung
- **IF <Bedingung> THEN... ELSE ...** - Verzweigung (muss in einer Zeile stehen)
- **INPUT <Dialogtext>; <Variable>** - Eingabe von Daten in eine Variable
- **INT (<Zahl>)** - Umwandlung einer Kommazahl in eine Ganzzahl
- **LOCATE <Zeile>, <Spalte>** - Positionierung des Cursors auf die angegebene Position
- **LPRINT** - Ausgabe auf den Drucker
- **ON <Variable> GOTO <Zeilennummer für Variable=1>, <Zeilennummer für variable=2>,...** - Mehrfachverzweigung abhängig vom Wert einer Variablen
- **PRINT <Variable> [; <Variable2>;...]** - Bildschirmausgabe von daten
- **REM** - Kommentar im Programm
- **+** - Addition
- **-** - Subtaktion
- **\*** - Multiplikation
- **/** - Division
- **^** - Potenzierung

## Anhang 2 – Die GW-BASIC-Entwicklungsumgebung

### Installation von GW-BASIC

GW-BASIC bzw. das funktionsidentische BASICA müssen nicht installiert werden. Die Entwicklungsumgebung besteht nur aus einer einzigen Datei namens GWBASIC.EXE oder BASICA.EXE. Zum Öffnen der Entwicklungsumgebung muss nur diese EXE-Datei gestartet werden.

### Aufruf von GW-BASIC und Kommandozeilenparameter

GW-BASIC wird mit folgendem Kommando aufgerufen:

```
GWBASIC [Programmname][<EinDatei] [>AusDatei] [/F:n] [/I] [/S:n] [/C:n]
[/M:[Adresse][,Blockgröße]] [/D]
```

Die einzelnen Kommandozeilenparameter haben die folgende Bedeutung:

- Programmname  
legt ein Quellspracheprogramm fest, das gleichzeitig mit GW-BASIC aufgerufen und ausgeführt wird. Ist keine Dateierweiterung angegeben, wird von einer .BAS-Datei ausgegangen.
- <EinDatei  
bewirkt, dass das Anwenderprogramm alle INPUT-, LINE INPUT-, INPUT\$- und INKEY\$-Eingaben aus der angegebenen Datei statt von der Tastatur holt
- >AusDatei  
gibt alle Programm Meldungen, Fehlermeldungen und PRINT-Ausgaben in die angegebene Datei anstatt auf den Bildschirm aus. Werden zwei statt einem Größenzeichen verwendet (also >>Ausdatei) so werden die Ausgaben an das Ende der Datei angehängt, ohne sie vorher zu löschen
- /F:n  
legt fest, wieviele Dateien gleichzeitig geöffnet werden können (Vorgabe: 3; Maximum: Wert von FILES in der CONFIG.SYS - 4 Dateien); der Parameter /I muss gleichzeitig gesetzt werden.
- /I  
reserviert den Speicherplatz, der durch die Parameter /F und /S zusätzlich benötigt wird, sofort nach dem Aufruf von GW-BASIC
- /S:n  
setzt die Größe des Datenpuffers und damit die maximale Satzlänge für Datensätze von Random-Dateien (Direktzugriffs-Dateien) fest; Vorgabe für n sind 128 Byte, maximal zulässig sind 32767 Byte
- /C:n  
legt die Größe des Empfangspuffers für die Datenübertragung mit seriellen Schnittstellen fest (der Standardwert für n ist 256 Byte pro Schnittstelle, Maximalwert: 32767 Byte, n=0 unterbindet die Kommunikation)
- [/M:[Adresse][,Blockgröße]]  
legt fest, wieviel Arbeitsspeicher BASIC nutzen kann und wie dieser Speicher aufgeteilt wird. BASIC verwaltet maximal und als Standard (wenn der Parameter nicht benutzt wird) insgesamt 64 KByte für Daten und Stack. Wenn Assembler Routinen geladen werden sollen, kann mit /M die höchste Speicheradresse benannt werden, die noch von BASIC benutzt werden darf; auf allen höheren Adressen können dann Anwenderprogramme (Assembler Routinen) geladen werden. Dadurch verkleinert sich der BASIC-eigene Speicherbereich. Dieser Parameter ermöglicht außerdem die Festlegung der maximalen Blockgröße, die BASIC verwaltet. Der Wert für die maximale Blockgröße errechnet sich aus der angegebenen "Blockgröße" (in Bytes) / 16.  
Beispiel: GWBASIC /M:32768,4096  
==> GW-BASIC benutzt mit diesem Parameter nur die ersten 32 KByte für BASIC. Die restlichen 32 KByte (ab Adresse 32769) werden für Assembler Routinen reserviert. Die maximale Blockgröße beträgt 4096/16 = 256 Byte.
- /D  
legt fest, dass die Funktionen ATN, COS, EXP, LOG, SIN, SQR und TAN mit doppelter Genauigkeit ausgeführt werden



## Allgemeine Bedienungshinweise für GW-BASIC

GW-BASIC ist nicht mit der Maus, sondern nur mit der Tastatur bedienbar. Du kannst nur die auf dem Bildschirm dargestellten Programmzeilen editieren. Der GW-BASIC-Editor arbeitet zeilenorientiert. Eine Scroll-Möglichkeit über den Bildschirminhalt hinaus ist nicht möglich. Daher muss sich der Programmierer mit LIST-Kommandos durch den Quellcode navigieren (siehe unten). Die Änderung einer Programmzeile muss durch die Eingabetaste abgeschlossen werden; der Cursor kann dabei an einer beliebiger Stelle in der Zeile stehen.

Je Zeile muss ein Befehl geschrieben werden oder mehrere durch Doppelpunkte voneinander getrennte Befehle. Vor jedem GW-BASIC-Befehl muss eine Zeilennummer stehen. Mit dem AUTO-Kommando können die Zeilennummern automatisch erzeugt und mit dem RENUM-Kommando neu angeordnet werden. Nachträglich eingefügte Befehlszeilen werden beim Listen des Programms automatisch an richtiger Stelle eingeordnet. Bestehende Befehlszeilen können gelöscht werden durch alleinige Eingabe der Zeilennummer, mit dem DELETE-Kommando oder mit der Esc-Taste.

Ein Programm muss nicht in der endgültigen Zeilenreihenfolge eingegeben werden. Die Zeilen werden automatisch richtig entsprechend ihrer Zeilennummer sortiert, und es können nachträglich Befehlszeilen angegeben werden. Das LIST-Kommando bewirkt eine Anzeige des Programms in der richtig sortierten Zeilen-Reihenfolge.

Am Bereitschaftszeichen (Ok) eingegebene Befehle ohne Zeilennummer werden bei Betätigen der Eingabetaste direkt abgearbeitet ("Direkt-Funktion"). Der Direktmodus eignet sich zum Testen von Befehlen oder für die Durchführung von Berechnungen.

## System- und Editierkommandos

Innerhalb der GW-BASIC-Entwicklungsumgebung werden Bedien-Kommandos entweder über die in der unteren Menüleiste aufgeführten Funktionstasten F1 - F10 oder durch Eintippen des Kommandos gestartet. Hier findest Du eine Liste der wichtigsten Kommandos, die auch "Systembefehle" oder "Systemkommandos" genannt werden. Optionale Syntaxelemente, die auch weggelassen werden können, sind in [eckige Klammern] gesetzt. Systemkommandos (außer den Editierkommandos) werden ohne Zeilennummer am Bereitschaftszeichen (Ok) angegeben und mit der Eingabetaste abgeschlossen.

- LOAD "<Pfadname><Dateiname>"  
Quellspachedatei \*.BAS in die Entwicklungsumgebung laden. Die Dateierweiterung .BAS kann weggelassen werden. Beachte, dass der Dateiname in Anführungszeichen gesetzt werden muss.
- LIST  
Zeigt das gesamte geladene Programm auf dem Bildschirm an
- LIST 10-200  
Zeigt die Programmzeilen 10 bis 200 an
- LIST -200  
Zeigt die Programmzeilen bis Zeile 200 an
- LIST 200-  
Zeigt die Programmzeilen ab 200 an
- RUN  
Startet das geladene Programm
- SYSTEM  
Beendet die GW-BASIC-Entwicklungsumgebung und bewirkt einen Rücksprung ins Betriebssystem MS-DOS
- <Strg+Pause>, <Strg+C> oder <Strg+Rollen>  
Unterbricht das laufende Programm, z.B. wenn es in einer Dauerschleife abgestürzt ist.
- SAVE "<Pfadname><Dateiname>" [,a]  
Quellspracheprogramm abspeichern. Ohne Parameter wird das Programm im eigenen GW-BASIC-Format abgespeichert. Der Parameter ",a" bewirkt, dass das Programm als normale ASCII-Textdatei abgespeichert wird, die auch von Textverarbeitungen, Editoren, QBasic usw. lesbar ist. Beachte, dass der Dateiname in Anführungszeichen gesetzt werden muss.
- CLS  
Löscht den Bildschirm

- DELETE <Zeilennummer>  
Löscht eine Zeile
- DELETE <1.Zeilennummer>--<letzte Zeilennummer>  
Löscht mehrere Zeilen
- Edit <Zeilennummer>  
Zeigt eine Zeile an und ermöglicht es, sie zu ändern. Die Änderung muss mit der Eingabetaste abgeschlossen werden. Der Cursor braucht dabei nicht an das Zeilenende geführt werden.
- <Einf>  
Schaltet beim Editieren zwischen Einfüge- und Ersetzenmodus um
- <Esc>  
Löscht die aktuelle Zeile
- <Strg+Ende>  
Löscht die aktuelle Zeile ab der Cursorposition.
- <Strg+Eingabetaste>  
Verschiebt den restlichen Text einer Zeile an den Anfang der nächsten Bildschirmzeile
- <Strg+Pos1>  
Löscht die Bildschirmanzeige und setzt den Cursor in die linke obere Ecke. Das eingegebene Programm bleibt erhalten.
- <Pos1>  
Setzt den Cursor an die linke obere Bildschirmecke
- <Ende>  
Setzt den Cursor an das Zeilenende
- <Strg+Druck>  
Schaltet das Ausdrucken aller Tasten-Eingaben ein und aus.
- AUTO <1. Zeilennummer> <Schrittweite>  
Aktiviert den automatischen Nummerierungsmodus für die folgenden Programmzeilen beginnend mit der 1. Zeilennummer und mit der angegebenen Schrittweite. Diese Automatik kann mit <Strg+C> abgeschaltet werden (siehe unten)
- <Strg+C>  
Schaltet den automatischen Nummerierungsmodus ab, z.B. den Direktmodus zu verwenden oder um Systembefehle einzugeben. ACHTUNG: <Strg+C> macht alle nicht gespeicherten Änderungen rückgängig!
- AUTO <1. Zeilennummer> (<Eingabetaste>) [<Eingabetaste>,...] <Strg+C>  
Erzeugt mit jeder Betätigung der Eingabetaste eine neue Zeilennummer für die Folgezeilen mit einer Schrittweite von 10.
- RENUM  
Nummeriert das Programm neu beginnend bei 10 in 10er-Schritten und passt alle Sprungzieleentsprechend an.
- RENUM 10000, 2000, 50  
Nummeriert alle Zeile ab Nummer 2000 neu. Die alte Zeile 2000 erhält die Nummer 10000. Die Schrittweite beträgt 50
- KEY OFF  
Schaltet die Funktionstastenanzeige in Zeile 25 aus
- KEY ON  
Schaltet die Funktionstastenanzeige in Zeile 25 aus
- KEY <Funktionstastennummer>, "<Befehl>" [+ CHR\$(13)]  
Belegt eine Funktionstaste mit einem häufig benutzten Befehl. Mit "+ CHR\$(13)" wird erreicht, dass nach dem Betätigen der Funktionstaste keine Eingabetaste gedrückt werden muss.
- NEW  
Löscht das aktuelle Programm im Arbeitsspeicher (nicht auf der Diskette/Festplatte) und ermöglicht das Eingeben oder Laden eines neuen Programms
- FILES "<Pfad>\"  
Zeigt eine Dateiliste des angegebenen Verzeichnisses / Ordners an
- KILL "<Pfadname><Dateiname>"  
Löscht die angegebene Datei
- NAME "<Pfadname><Dateiname>" AS "<Pfadname><Dateiname>"  
Benennt eine Datei um
- SHELL  
Wechselt vorübergehend zum MS-DOS-Betriebssystem.
- EXIT  
Bewirkt eine Rückkehr von dem vorübergehend per SHELL geöffneten Betriebssystem zur GW-BASIC-Entwicklungsumgebung.

## **Anhang 3 - Weitere Informationen zu GW-BASIC**

- Ein komplettes englischsprachiges Handbuch zu GW-BASIC gibt es auf meiner Webseite [www-qbasic.de](http://www-qbasic.de) in der Rubrik "QBasic | E-Books"
- Etliche Lernkurse und Tutorials zu GW-BASIC gibt es auf meiner Webseite [www-qbasic.de](http://www-qbasic.de) in der Rubrik "QBasic | Tutorials | GW-BASIC"
- Eine Riesenauswahl an Buchbesprechungen zu GW-BASIC gibt es auf meiner Webseite [www-qbasic.de](http://www-qbasic.de) in der Rubrik "QBasic | Bücher | GW-BASIC"
- Jede Menge Downloads und Tutorials zu GW-BASIC gibt es [www.gwbasic.at](http://www.gwbasic.at)