Personal Computer

GW™-BASIC

GW (GW™-BASIC) ist ein Warenzeichen der Microsoft Corporation. IBM ist ein eingetragenes Warenzeichen der International Business Machines Corporation.

1. Auflage, September 1984

NCR ist ständig bemüht, die Produkte im Zuge der Entwicklung von Technologie, Bauteilen, Soft- und Firmware dem neuesten Stand anzupassen. NCR behält sich deshalb das Recht vor, Spezifikationen ohne vorherige Ankündigung zu ändern.

Nicht alle hier beschriebenen Leistungen werden von NCR in allen Teilen der Welt vertrieben. Nähere Informationen bezüglich eventueller Einschränkungen oder auch Erweiterungen sowie den aktuellen Stand erfahren Sie von Ihrem Händler oder der nächstgelegenen NCR-Geschäftsstelle.

GW-BASIC

Vorwort

GW-BASIC ist eine weitverbreitete Programmiersprache, die zur optimalen Nutzung der Leistungen Ihres NCR Personal Computer beiträgt.

Diese beinhalten Diskettenzugriff, Drucken, Datenkommunikation, hochauflösende Graphik und sogar Musik.

Mit Hilfe des bildschirmorientierten Editors und dem umfassenden GW-BASIC-Befehlsregister können Sie Programme für einen breiten Anwendungsbereich entwickeln: die vielseitigen Druck- und Zeicheninstruktionen ermöglichen mit dem Programm das Erstellen, Speichern und Abrufen von Listen, Briefen und Geschäftsgrafiken, die sich ganz nach Ihren Anforderungen richten.

Zusätzlich steht Ihnen eine große Anzahl von mathematischen Funktionen zur Verfügung. Wenn Sie Spaß am Progammieren haben, können Sie auch etwas „Musik" in Ihre Programme aufnehmen.

GW-BASIC stellt Ihnen außerdem Programmierhilfen zur Verfügung, die es Ihnen ermöglichen, den Farbbildschirm, einen Lichtgriffel und einen joystick (Steuerknüppel) voll zu nutzen.

Kapitel 1 bietet sowohl dem Anfänger als auch dem erfahrenen Programmierer Anleitungen für das Starten und Verlassen von GW-BASIC nach Bearbeitung oder Ablauf des Programms. Das Kapitel beschreibt weiter, welche Meldungen GW-BASIC am Bildschirm ausgibt, und wie es von Ihnen eingegebene Informationen speichert. Die letzten Abschnitte des ersten Kapitels geben Ihnen schließlich einen Einblick in die Entscheidungen, die ein GW-BASIC-Programm treffen kann, sowie die Hilfe, die es von Ihnen dafür braucht. Als Hilfestellung für den Anfänger ist die Einleitung von Beispielen und Übungen begleitet.

Kapitel 2 gibt einen Überblick über den GW-BASIC Programm-Editor und beschreibt die Leistungen, die GW-BASIC für das Erstellen von neuen und das Ändern von bestehenden Programmen, besitzt. Wenn Sie schon mit einem Zeileneditor, wie er in vielen BASIC-Versionen angeboten wird, gearbeitet haben, werden Sie die Fähigkeiten von GW-BASIC, hinsichtlich der bildschirmorientierten Textverarbeitung, besonders zu schätzen wissen.

Kapitel 3 stellt mit den vielfältigen Möglichkeiten der Bildschirmsteuerung die herausragende Fähigkeit von GW-BASIC vor. Zusätzlich zum erweiterten Zeichensatz, der Ihnen auf einem Adapter mit einfarbigem Bildschirm zur Verfügung steht, haben Sie besonders bei einem Adapter mit Farbbildschirm die Möglichkeit, die Graphik voll auszuschöpfen. Wie in Kapitel 1, sind diese Beschreibungen wieder durch Beispiele und Übungen ergänzt.

Die einleitenden Seiten von Kapitel 4 führen Ihnen eine Zusammenstellung der gesamten GW-BASIC-Befehle auf. Viele der Bezeichnungen sind selbsterklärend, so hat zum Beispiel BEEP ganz offensichtlich mit Lauterzeugung zu tun, CIRCLE zeichnet genau diese geometrische Figur. Die Zusammenstellung ist aufgeteilt in Abschnitte, von denen jeder einen bestimmten Gesichtspunkt der Programmierung behandelt, zum Beispiel „Das Laden und Speichern von Programmen" und „Der Lautsprecher". Dann folgt, in alphabetischer Reihenfolge, eine ausführliche Beschreibung jedes Kommandos und jeder Funktion des GW-BASIC-Befehlsregisters mit vielen Programmierbeispielen. Daher dient dieses Kapitel sowohl als Nachschlagewerk für erfahrene Programmierer als auch als nützlicher Wegweiser für Programmieranfänger. Selbst wenn ein Beispiel Ihren individuellen Programmieransprüchen einmal nicht genügen sollte, finden Sie dort die Informationen, die Ihnen helfen, das Beispiel so abzuändern, um das gewünschte Ergebnis zu erzielen.

In Kapitel 5 werden die verschiedenen Arten der Diskettenspeicherung und die Funktionsweise des Datenaustausches von GW-BASIC mit Peripheriegeräten, wie zum Beispiel einem Drucker, beschrieben. Vielleicht sind Sie mit Bezeichnungen wie „wahlfreier Zugriff" und „sequentieller Zugriff" im Zusammenhang mit Dateien schon vertraut, wenn nicht, so helfen Ihnen zahlreiche Erläuterungen und Beispiele weiter.

Kapitel 6 und 7 sind hauptsächlich für Assembler-Programmierer gedacht, die Maschinensprachroutinen in ein GW-BASIC-Programm einbauen möchten, oder daran interessiert sind, wie GW-BASIC den Speicher des NCR Personal Computer nutzt.

GW-BASIC ist zum Testen von Programmen ideal geeignet. Die Befehle RUN und GOTO ermöglichen Ihnen den Zugriff auf ein Programm an jeder beliebigen Stelle. Das STOP-Kommando bewirkt genau das, was sein Name besagt, und eine spezielle Sucheinrichtung (TRON/TROFF) ermöglicht eine Übersicht über den Ablauf Ihres jeweiligen Programms. Außerdem ist GW-BASIC in der Lage, Ihnen wichtige Fehlerhinweise bei Programmstörung zu geben. Wenn Sie GW-BASIC beispielsweise anweisen, die Note K zu spielen, wird es Ihnen nicht nur mitteilen, daß es diese Note nicht gibt, sondern auch, wo die falsche Anweisung im Programm aufgetreten ist.

Als erfahrener Programmierer möchten Sie die vielfältigen Möglichkeiten von GW-BASIC bei Ihrer Arbeit sicher bald nicht mehr missen. Doch auch dem Anfänger bietet dieses Handbuch zahlreiche Informationen, Beispiele und Übungen, die ihm, zusammen mit den eingebauten Hilfsfunktionen von GW-BASIC auf seinem Weg zum geübten Programmanwender behilflich sein werden.

GW-BASIC

Inhaltsverzeichnis

Kapitel 5

Einführung in GW-BASIC

Das Programm GW-BASIC befindet sich auf Ihrer NCR-DOS Diskette. Es läßt sich unter den Namen GWBASIC, GW-BASIC, BASIC oder BASICA aufrufen. Welchen Namen Sie auch verwenden, es wird immer das Programm GW-BASIC geladen. Das Programm selbst ist nur einmal vorhanden. Es kann aber unter jedem der vier Namen erreicht werden. Dies ist erforderlich, weil Anwenderprogramme nicht immer einheitlich den Namen GW-BASIC verwenden, sondern auch die anderen Namen.

Diese vier Namen sind als reine Lesedateien gekennzeichnet, damit sie nicht unabsichtlich gelöscht werden können. Wenn Sie mit DISKCOPY eine Sicherungskopie gemacht haben, dann lassen sich die vier Einträge nicht aus dem Katalog der Zieldiskette löschen. Wenn Sie eine Diskette erzeugen wollen, die nicht alle vier Einträge aufweist, dann benutzen Sie bitte den COPY-Befehl. Geben Sie jedoch den Befehl COPY *.* ein, versucht das System vier Kopien des Programms GW-BASIC zu erstellen, nämlich eine für jeden Eintrag im Katalog. Weil aber auf der Zieldiskette nicht genug Platz für die zusätzlichen Dateien ist, erscheint die Meldung:

Disketten- /Plattenplatz nicht ausreichend

Zu diesem Zeitpunkt sind alle Dateien - einschließlich GWBASIC - übertragen worden. Es fehlen die Dateien GW-BASIC, BASIC und BASICA, die als letzte im Katalog aufgeführt sind.

Bevor Sie beginnen, sollten Sie schon einige grundlegende Vorgänge im NCR-DOS-Betriebssystem kennen, zum Beispiel, wie man Disketten kopiert und Befehle an das Betriebssystem erteilt. Es ist weiterhin hilfreich, wenn Sie schon mit der Vergabe von Dateibezeichnungen vertraut sind. Es lohnt sich sicherlich für Sie, diese Einzelheiten Ihres Betriebssystems kennenzulernen, bevor Sie beginnen, mit GW-BASIC zu arbeiten.

Das Starten von GW-BASIC

Zuerst laden Sie das NCR-DOS-Betriebssystem in den Speicher Ihres Computers. Wenn Sie diesen Vorgang noch nicht kennen, sollten Sie in Ihrem "NCR-DOS-Handbuch" nachschlagen. Sobald das Systembereitschaftszeichen (system prompt) am Bildschirm erscheint, können Sie GW-BASIC in den Speicher laden. Geben Sie nun einen der vier Namen ein, zum Beispiel:

GWBASIC

Denken Sie bitte daran, daß ein Befehl nur dann vollständig ist, wenn Sie anschließend die Taste <CR> betätigen. Wenn GW-BASIC erfolgreich geladen wurde, wird es seine Bereitschaft durch

GW-BASIC 2.01
© Copyright Microsoft 1983, 1984
The NCR Personal Computer Model 4
Version 2.01.00
Copyright NCR Corp., 1984
38021 Bytes free
Ok

anzeigen. „Ok" bedeutet, daß GW-BASIC bereit ist, Ihre Befehle entgegenzunehmen. Die Anzeige am unteren Bildschirmende bezieht sich auf die Funktionstasten der Tastatur, und ist für Sie momentan noch nicht von Bedeutung.

Dem erfahreneren Programmierer stehen bei GW-BASIC weitere Lademöglichkeiten zur Verfügung. Wenn GW-BASIC also neu für Sie ist, können Sie die nun folgenden Seiten bis zum Abschnitt „Das Verlassen von GW-BASIC", überspringen.

Für zusätzliche Möglichkeiten, GW-BASIC zu laden, benötigen Sie wahlweise weitere Angaben im Ladebefehl:

GWBASIC Dateispezifikation

Dateispezifikation steht hierbei für die Bezeichnung einer Programmdatei (steht nicht in Klammern). Sie teilt GW-BASIC mit, diese Datei zu finden, sie zu laden und zu starten (RUN), ohne auf weitere Anweisungen von Ihnen zu warten, und ohne Anzeige der Copyright-Mitteilung. Die Dateispezifikation muß den NCR-DOS Dateisspezifikationsrichtlinien entsprechen. Er kann außerdem auch einen Pfad bezeichnen. Wenn Sie keinen Namenszusatz angeben, der Dateiname aber kürzer ist als 9 Zeichen, nimmt GW-BASIC automatisch an, daß der Namenszusatz .BAS ist. Beispiele:

GWBASIC ALTPROG.ABC

lädt GW-BASIC in den Speicher und startet (RUN) das Programm der Datei ALTPROG.ABC, wohingegen

GWBASIC ALTPROG

GW-BASIC in den Speicher lädt, und das Programm der Datei ALTPROG.BAS startet. Es ist für Sie von Vorteil, Ihren GW-BASIC-Programmen den Namenszusatz .BAS zu geben, da Sie sich beim Betrachten Ihres Diskettenverzeichnisses leichter unterscheiden lassen. Wenn

Sie beim Laden von NCR-DOS die AUTOEXEC.BAT-Datei für die automatische Ausführung einer Folge von GW-BASIC-Programmen benutzen, sollten Sie jedes Programm mit dem GW-BASIC SYSTEM-Befehl abschließen, um die Ausführung des nächsten Befehls in der AUTOEXEC.BAT-Datei sicherzustellen.

<stdin

bezieht sich auf das Standard-Eingabegerät, normalerweise ist das die Tastatur. stdin ist die Bezeichnung einer Datei, von der GW-BASIC stattdessen Daten entgegennimmt. Entscheiden Sie sich für diese wahlfreie Angabe,so muß sie jeder anderen Angabe, die mit einem Schrägstrich (/) beginnt, vorangestellt werden.

>stdaus

bezieht sich auf das Standard-Ausgabegerät, normalerweise ist das der Bildschirm. stdout steht für den Namen einer Datei, an die GW-BASIC Daten ausgibt. Sollten Sie sich zu dieser wahlfreien Angabe entschließen, muß sie jeder anderen, mit einem Schrägstrich (/) beginnenden Angabe, vorangestellt werden. Wird '>>' anstelle von '>' verwendet, wird die Ausgabe an die Datei gehängt, andernfalls wird die Datei überschrieben.

GWBASIC/F:n

Hier steht „n" für die Anzahl von Plattendateien(maximal 15), die zu einem beliebigem Zeitpunkt während eines Programmablaufs geöffnet sein können. Beispiel:

GWBASIC/F:6

zeigt GW-BASIC an, daß bis zu sechs Dateien geöffnet sein können. Jede Datei hat einen Speicherbedarf von 62 Byte für den Dateisteuerblock und die Pufferkapazität (siehe Wahlmöglichkeit /S). Die Zahl der zu einem Zeitpunkt geöffneten Dateien hängt ab vom Wert, der dem FILES-Parameter der Betriebssystem-Datei CONFIG.SYS zugeteilt ist. Der Standardwert des Parameters ist 8. GW-BASIC selbst benötigt vier Dateien. Hat CONFIG.SYS FILES=8 festgelegt, dann liegt der Maximalwert für die Dateienanzahl in der Wahlmöglichkeit /F bei 4. Nachdem sämtliche Dateien geöffnet wurden, werden alle Versuche, weitere Dateien zu öffnen, mit der Fehlermeldung „Zu viele Dateien" beantwortet.

GWBASIC /M:Adresse, Blockgröße

In diesem Beispiel entspricht die Adresse dem Speicherplatz mit der

höchstmöglichen Speicheradresse der von GW-BASIC benutzt werden kann. Diese Wahlmöglichkeit ist hilfreich bei der Reservierung von Raum für Ihr Programm im oberen Speicherbereich. Der Wert für die Adresse sollte natürlich sinnvoll sein, d. h., es sollte zumindest genug Platz für Ihr Programm vorhanden sein. Der maximale reservierbare Speicherplatz beträgt 64 KB. Wenn keine Eingabe gemacht bzw. 0 eingegeben wird, versucht GW-BASIC, so viel wie möglich bis zu maximal 65536 Byte zuzuordnen. Beispiel:

GWBASIC /M:32768

ermöglicht GW-BASIC die Nutzung der ersten 32 KB des vom Betriebssystem zugeteilten Programmsegments. Den zweiten Teil der wahlfreien Eingabe können Sie für die Zuteilung einer maximalen Blockgröße verwenden. Dies empfiehlt sich, wenn Sie Programme oberhalb der im ersten Teil der wahlfreien Angabe definierten Adresse laden möchten. Bei Verwendung des „SHELL"-Befehls muß die Blockgröße bereits definiert worden sein. Andernfalls wird das aufrufende Programm von dem mit „SHELL" aufgerufenen Programm überschrieben. Unter Blockgröße versteht man die Anzahl von Speicherparagraphen (je 16 Byte), die als Arbeitsplatz für GW-BASIC benötigt werden, und den zusätzlichen Speicherplatz, der außerhalb des GW-BASIC-Programmbereiches nötig ist. Beispiel:

GWBASIC /M:32000,2048

In diesem Beispiel ergibt sich eine Summe von 32768 reservierten Bytes (2048 x 16), von denen 32000 für GW-BASIC, und 768 für die Benutzung außerhalb von GW-BASIC zur Verfügung stehen.

GWBASIC /S:Größe

Hier wird die Größe des Puffers für die Bearbeitung von Dateien mit wahlfreiem Zugriff bestimmt. Der Maximalwert ist 32767. Diese Wahlmöglichkeit bestimmt die maximale Datensatzlänge von Datensätzen, die vom OPEN-Befehl festgelegt werden können.

GWBASIC /C:ÜbertrGröß

legt die Größe des für den Datenempfang benutzten Puffers bei asynchroner Datenübertragung fest. Machen Sie von dieser Möglichkeit keinen Gebrauch, so tritt die wahlfreie Angabe außer Kraft. Der gültige Maximalwert für diese Angabe ist 32767. Wenn Sie für diese den Wert 0 setzen, wird die RS232C-Schnittstelle abgeschaltet, mit dem Ergebnis, daß der Pufferplatz nicht benötigt wird. Dies hat zur Folge, daß Teile von GW-BASIC, die für die Datenübertragung erforderlich sind, nicht von der Diskette geladen werden.

Bei Benutzung von zwei asynchronen Datenschnittstellen gilt Übertr-Größ für beide. Der empfohlene Wert für eine Hochgeschwindigkeitsdatenleitung beträgt 1024. Der Puffer für die Datenübertragung liegt immer bei 128 Bytes.

/D

Dieser wählbare Zusatz teilt GW-BASIC mit, ob Sie bei den folgenden mathematischen Funktionen mit „doppelter Genauigkeit" arbeiten wollen: ATN, COS, EXP, LOG, SIN, SQR, und TAN. Die Einbeziehung dieser Wahlmöglichkeit erfordert ungefähr 3000 Bytes mehr an Speicherplatz, als sonst von GW-BASIC eingenommen würde.
Wenn Sie für diese wahlfreien Eingaben keine Werte angeben, setzt GW-BASIC sogenannte Standardwerte: die maximale Anzahl von Dateien, die zu einem Zeitpunkt geöffnet sein können, ist hier 3, die Benutzung des Speichers ist nicht eingeschränkt, die Datensatzlänge beträgt 128 Bytes, die Puffergröße für asynchrone Datenübertragung ist 256 Bytes, die /D-Option ist dabei ausgeschaltet.

Beim Laden von GW-BASIC können diese Optionen zu einem Befehl zusammengefaßt werden. Hier einige Beispiele:

GWBASIC GEHALT.BAS /F:6

GW-BASIC wird geladen, und die Programmdatei GEHALT.-BAS ausgeführt. Dabei dürfen bis zu 6 Dateien zu einem Zeitpunkt geöffnet sein.

GWBASIC /M:32768

GW-BASIC wird geladen und wartet auf weitere Anweisungen. Speicherplatz oberhalb von 32768 steht GW-BASIC nicht zur Verfügung.

GWBASIC DATEN /F:2/M:32768

GW-BASIC wird geladen, und die Programmdatei GWBASIC DATEN.BAS ausgeführt. Zu einem Zeitpunkt dürfen nicht mehr als 2 Dateien geöffnet sein. Speicherplatz oberhalb von 32768 steht GW-BASIC nicht zur Verfügung.

RÜCKKEHR ZUM NCR-DOS BETRIEBSSYSTEM

Durch Eingabe des Befehls

SYSTEM

verlassen Sie GW-BASIC und kehren zur NCR-DOS Betriebssystemebene zurück. Beachten Sie jedoch, daß Sie durch Betätigen der Control-

Break Taste nicht auf die NCR-DOS-Ebene zurückkehren können. Diese Taste dient zum Abbruch eines GW-BASIC-Programms, um wieder zur GW-BASIC-Ebene zurückzukehren. Dies wird durch das Erscheinen von „Ok" am Bildschirm bestätigt.

SPEICHERN UND LADEN EINES PROGRAMMS

Wenn Sie sich noch nicht mit GW-BASIC beschäftigt haben, sollten Sie diesen Abschnitt vor Ausführung der folgenden Übungen lesen.

Sobald Sie GW-BASIC geladen haben, erscheint „Ok" am Bildschirm. GW-BASIC teilt Ihnen dadurch mit, daß es auf die Eingabe Ihrer Befehle wartet. Jetzt können Sie mit dem Schreiben Ihres Programms beginnen. Programm zu schreiben. Bevor Sie jedoch Ihr Programm testen, ist es zweckmäßig, das Programm auf Platte zu speichern. Dafür benötigen Sie den SAVE-Befehl. Hier müssen Sie jetzt entscheiden, welchen Namen Sie Ihrem Programm geben wollen. Im folgenden Beispiel wird ein Programm unter dem Namen NEWPROG.BAS auf der augenblicklich aktivierten Platte gespeichert:

```
SAVE „NEUPROG"
```

Dieser Befehl kann immer dann eingegeben werden, wenn „Ok" als letzte Meldung am Bildschirm erscheint oder sobald der blinkende Cursor anzeigt, daß GW-BASIC auf die Eingabe einer weiteren Programmzeile wartet. Bei erneuter Ausgabe der „Ok"-Meldung ist Ihr Programm auf Platte gespeichert. GW-BASIC nimmt an, daß das, was Sie auf Platte gespeichert haben, ein GW-BASIC-Programm ist. Der Name einer Programmdatei erhält dann die Erweiterung BAS. Sie können auch den Namenszusatz BAS. (SAVE „NEUPROG.BAS") selbst bestimmen, das Ergebnis bleibt davon unberührt. Wenn Sie es wünschen, können Sie auch einen anderen Namenszusatz für Ihren Dateinamen festlegen. GW-BASIC wird dies akzeptieren jedoch jedesmal, wenn Sie mit Ihrem Programm arbeiten wollen, müßten Sie dann sowohl den Namenszusatz als auch den Dateinamen eingeben. Daher erscheint es wesentlich sinnvoller, wenn GW-BASIC selbst den Namenszusatz .BAS dem Dateinamen gibt.

Wollen Sie zu einem späteren Zeitpunkt Ihr Programm ändern bzw. erweitern, laden Sie zunächst GW-BASIC. Bei Ausgabe der „Ok"-Meldung geben Sie

```
LOAD „NEUPROG"
```

ein und Ihr Programm wird wieder in den Arbeitsspeicher übertragen.

Jetzt können Sie Ihr Programm weiterschreiben (editieren) bzw. unter Verwendung des RUN-Befehls ablaufen lassen. Durch Eingabe von

RUN

können Sie das Programm durchführen. Wissen Sie schon von vornherein, daß Sie ein bereits vorhandenes Programm ohne weitere Editierung ablaufen lassen wollen, können Sie zwischen zwei anderen Methoden wählen. Zeigt Ihr NCR-PC die Bereitschaft des NCR-DOS-Betriebssystems an, können Sie den Programmnamen (ohne Anführungszeichen) angeben, sobald Sie GW-BASIC laden:

GWBASIC NEUPROG

Haben Sie nach der zweiten Methode GW-BASIC geladen und wurde die Copyright-Mitteilung und das „Ok" am Bildschirm ausgegeben, können Sie nun

RUN „NEUPROG"

eingeben.

In beiden Fällen wird dann das Programm ohne Ihr Zutun ausgeführt. Es ist wenig sinnvoll, diese Befehle vorerst einzugeben, da Sie noch kein Programm für GW-BASIC auf Platte gespeichert haben. Haben Sie jedoch gerade versucht, ein nicht vorhandenes Programm zu laden bzw. ablaufen zu lassen, werden Sie bemerken, daß GW-BASIC beanstandet, daß es die Datei mit dem Programm nicht gefunden hat. Eine solche Fehlermeldung ist jedoch kein Grund zur Besorgnis. Auf diese Weise macht Sie GW-BASIC darauf aufmerksam, welche Information fehlt bzw. es zeigt Ihnen an, wo Sie etwas falsch gemacht haben.

ÜBUNGEN

Stellen Sie das System durch Betätigen von Control-Alt-Del zurück und laden Sie das NCR-DOS-Betriebssystem. Durch Eingabe von

GWBASIC

auf der NCR-DOS-Betriebsebene wird GW-BASIC geladen. GW-BASIC meldet dann seine Bereitschaft mit „Ok". Geben Sie nun folgende Zeile ein:

20 PRINT „So kurz kann ein Programm sein"

Diesmal meldet sich GW-BASIC nicht mit „Ok", sondern wartet darauf, daß Sie weitere Programmzeilen eingeben. Es kann sich hier um ein sehr

kurzes Programm handeln, das jedoch auf Platte gespeichert werden kann. Geben Sie diesem Programm einen Namen, z.B. MINIPROG, und geben Sie folgenden GW-BASIC-Befehl ein (falls Sie mit den Regeln der NCR-DOS-Benennung von Dateien nicht vertraut sind, geben Sie Ihrer Datei einem Namen bis zu 8 Buchstaben)

SAVE „MINIPROG"

Meldet sich GW-BASIC wieder mit „Ok", so verlassen Sie GW-BASIC und kehren durch Eingabe von

SYSTEM

wieder zur Betriebssystemebene zurück. Durch Erteilen des NCR-DOS DIR-Befehls können Sie jetzt prüfen, ob MINIPROG.BAS in das Inhaltsverzeichnis der Platte eingetragen wurde.
Sie können jetzt die Arbeit an Ihrem Programm wieder aufnehmen, indem Sie GW-BASIC zuerst laden und dann

LOAD „MINIPROG"

eingeben. Die „Ok"-Meldung am Bildschirm teilt Ihnen mit, daß Ihr Programm gefunden und geladen wurde. Sie können den Inhalt Ihres Programms durch Eingabe von

LIST

überprüfen, worauf auf dem Bildschirm die Programmzeile gefolgt von „Ok" ausgegeben wird. Fügen Sie nun dem Programm an der Stelle, wo sich der blinkende Cursor befindet, folgende Eingabe hinzu:

10 CLS

Dann geben Sie nochmals LIST ein. Sie werden wahrscheinlich bemerkt haben, daß GW-BASIC die beiden Programmanweisungen in die „richtige Reihenfolge" gebracht hat, d.h. die niedrigere der beiden Zahlen steht an erster Stelle. In dieser Reihenfolge werden die Programmanweisungen ausgeführt, sobald RUN eingegeben wird. Zunächst jedoch ist es zweckmäßig, die neu geänderte Version des Programms auf die gleiche Art wie oben zu speichern, und zwar durch Eingabe von:

SAVE „MINIPROG"

Für einen Versuchsprogrammablauf geben Sie einfach

RUN

ein. Der erste der beiden Befehle (mit der vorangestellten Zahl 10) löscht den Inhalt des Bildschirms, während der zweite Befehl den Text zwischen den Anführungsstrichen am Bildschirm anzeigt (PRINT). Die Ausführung dieses kurzen Programms erfolgt sehr schnell, worauf sich GW-BASIC wieder mit „Ok" meldet. Kehren Sie jetzt wieder zur Betriebsebene (SYSTEM) zurück, damit Sie eine weitere Methode für den Programmablauf versuchen können. Sobald die NCR-DOS-Betriebsbereitschaft erscheint, geben Sie

GWBASIC MINIPROG

ein. Dabei wird zunächst GW-BASIC und unmittelbar darauf MINIPROG geladen. MINIPROG wird dann ohne weitere Anweisungen Ihrerseits ausgeführt.

Falls Sie bei der Eingabe einer Programmzeile (Befehl mit vorangestellter Nummer) einen Fehler machen, geben Sie einfach die Zeile in ihrer unvollständigen bzw. falschen Form ein und schreiben Sie die Zeile in der richtigen Form unter Verwendung der gleichen Zeilennummer. Die alte falsche Version wird durch die richtige ersetzt, sobald die richtige Version eingegeben ist. Falls Sie einen Fehler übersehen, der gegen die Sprache von GW-BASIC verstößt, wird er erst beim Ablauf des Programms von GW-BASIC festgestellt. GW-BASIC hält dann das Programm an und zeigt Ihnen die Zeilennummer, wo sich der Fehler befindet. Um eine Programmzeile ersatzlos löschen zu können, geben Sie einfach DELETE zusammen mit der betreffenden Zeilennummer ein. Die gesamten Leistungen des Editors werden im Kapitel „Bildschirmeditor" (Kapitel 2) beschrieben.

Die Zeilennummern geben GW-BASIC die Reihenfolge an, in der Sie die Ausführung der Befehle wünschen. Eine Zeilennummer muß sich im Zahlenbereich 0 bis 65529 bewegen; es besteht jedoch kein Grund dafür, warum ein Programm von 5 Zeilen die Zeilennummern 0,1,2,3 und 4 verwenden muß. Dies wäre tatsächlich nicht wünschenswert, da es ein späteres Einfügen zusätzlicher Programmzeilen verhindern würde. Daher ist es besonders von Nutzen, wenn Sie Ihre Programme mit einem Zeilenabstand von 10 schreiben. Der RENUM-Befehl bietet eine Möglichkeit für mehr Platz zwischen den jeweiligen Programmzeilen, falls so etwas überhaupt nötig sein sollte. AUTO ist ein Befehl, der Ihnen Zeilennummern mit Abständen anbietet, die von Ihnen selbst festgelegt wurden. Somit ersparen Sie sich die Mühe, für jede einzelne Programmzeile die Zeilennummer eingeben zu müssen.

GW-BASIC verwendet Kurzformen für die Befehle EDIT, LIST, AUTO und DELETE, die sich auf die gerade bearbeitete Zeile, nämlich den Punkt (.), beziehen. Wenn Sie daher

LIST .

eingeben, wird die von Ihnen augenblicklich bearbeitete Programmzeile am Bildschirm ausgegeben.
Die beiden GW-BASIC Zeilen, die das Programm MINIPROG.BAS umfassen, bestehen jeweils nur aus einem Befehl. In GW-BASIC können Programmzeilen mit mehr als einen Befehl vorkommen, in einem solchen Fall müssen die einzelnen Befehle durch Kommas voneinander getrennt werden. Zum Beispiel hätte die folgende Zeile die gleiche Wirkung wie die beiden Zeilen von MINIPROG.BAS zusammen:

10 CLS:PRINT „So kurz kann ein Programm sein"

Bei der Eingabe einer Programmzeile ist eine Überschreitung der am Bildschirm ausgegebenen Programmzeile sogar zulässig, vorausgesetzt die Programmzeile ist nicht länger als 254 Zeichen mit abschließender Betätigung von <CR>.

BETRIEBSARTEN

Beim Laden vom GW-BASIC wird die „Ok"-Meldung ausgegeben, die anzeigt, daß GW-BASIC bereit ist, Ihre Befehle anzunehmen. An dieser Stelle können Sie zwei Betriebsarten wählen: die indirekte und die direkte Betriebsart.

Bei der indirekten Betriebsart geben Sie die Programmzeilen so ein, wie sie es in den vorausgegangenen Übungen praktiziert haben. Man bezeichnet diese Betriebsart indirekt, weil die Befehle erst dann ausgeführt werden, wenn der Befehl für den Ablauf des Programms erteilt wird. Sie brauchen GW-BASIC nicht mitzuteilen, daß Sie diese Betriebsart benötigen: denn sobald nämlich diesem Befehl eine Zeilennummer vorausgeht, erkennt GW-BASIC, daß der Befehl nicht für eine unmittelbare Ausführung bestimmt ist, sondern Teil eines Programms ist, das zu einem späteren Zeitpunkt ausgeführt, gespeichert und und wiedergefunden werden kann.

Bei den Befehlen LOAD, SAVE, RUN und SYSTEM, wie sie in den vorausgegangenen Übungen verwendet wurden, handelt es sich um direkte Befehle, das bedeutet, daß sie einen unmittelbaren Effekt haben. Durch die Verwendung von GW-BASIC-Befehlen als direkte Befehle läßt sich leicht prüfen, was in einem Programm vorgegangen ist. Dies macht es Ihnen möglich, GW-BASIC und Ihren Computer als Rechner für schnell durch-führbare Rechenvorgänge zu verwenden. Versuchen Sie einmal, folgenden einfachen Rechenvorgang als direkten Befehl einzugeben:

PRINT 128+64-5

Das Ergebnis wird sofort am Bildschirm ausgegeben. Der Verzicht auf Zeilennummern, um sofortige Ergebnisse zu erzielen, besagt daß der Befehl nicht über seine Ausführung hinaus gespeichert wird (obgleich das Ergebnis das gleiche ist, als wenn er als Teil eines Programms ausgeführt worden wäre). Wenn daher ein Befehl bzw. eine Folge von Befehlen wiederholt verwendet werden soll, ist es am zweckmäßigsten, ihnen Zeilennummern voranzustellen und sie auf Platte zu speichern. Aus diesem Grund werden Computerprogramme erstellt.

DER ZEICHENSATZ

Der GW-BASIC Zeichensatz umfaßt alle Buchstaben des Alphabets sowie numerische Zeichen (die Ziffern 0 bis 9). Zusätzlich gehört eine Anzahl von Sonderzeichen zum GW-Zeichensatz. Man kann erkennen, daß einige von ihnen arithmetische Funktionen bezeichnen. Weitere Sonderzeichen sind beim Programmieren mit GW-BASIC von besonderer Bedeutung. Diese Zeichen werden in den entsprechenden Abschnitten dieses Handbuches erklärt. Im folgenden finden Sie eine Liste der Sonderzeichen

Beschreibung	Symbol	Bedeutung in GW-BASIC
Leerstelle		Trennt Syntax-Elemente in einer Programmzeile
Gleichheitszeichen	=	Übliche Funktion bei arithmetischen Vergleichsoperationen Teilt einer Programmvariablen Werte zu
Pluszeichen	+	Übliche arithmetische Bedeutung, Textverkettung
Minuszeichen	−	Übliche arithmetische Bedeutung
Sternchen	*	Multiplikationssymbol
Schrägstrich	/	Divisionssymbol
Pfeil nach oben	∧	Potenzierungssymbol
Linke und rechte Klammer	( )	Übliche algebraische Bedeutung
Prozentzeichen	%	Kennzeichnet eine ganze Zahl

Doppelkreuz	#	Kennzeichnet eine Zahl mit doppelter Genauigkeit
Ausrufezeichen	!	Kennzeichnet eine Zahl mit einfacher Genauigkeit
Dollarzeichen	$	Kennzeichnet einen Text
Komma	,	Verwendet bei der Bildschirm- und Druckerformatierung
Semikolon	;	
Punkt	.	Dezimalstelle
Einfaches Anführungszeichen	'	Begrenzt eine Bemerkung des Programmierers
Doppeltes Anführungszeichen	"	Begrenzt einen Text
Doppelpunkt	:	Trennt Befehle innerhalb einer Programmzeile
Und-Zeichen	&	Verwendet zur Festlegung des Zahlenmodus
Fragezeichen PRINT-Befehl	?	Abkürzung des Editors für den
Weniger als Größer als	< >	Übliche algebraische Bedeutung
Schrägstrich nach links	\	Symbol für Ganzzahldivision

GW-BASIC erkennt auch eine Anzahl von Kombinationen der Control-Taste für die Programmeditierung. Solche Kombinationen werden in dem Abschnitt, der sich mit dem GW-BASIC-Bildschirm-Editor befaßt, beschrieben.

Die GW-BASIC Zeichenkette stellt eine Erweiterung des allgemein bekannten und benutzten ASCII-Code dar. Der ASCII-Code teilt 128 Einzelzeichen einen Wert zu. Das ASCII-Codezeichen für den Großbuchstaben A ist 65, und 51 für Ziffer 3. Die sogenannten „Control"-Steuerzei-

chen, d.h. solche Codes, die nicht direkt am Bildschirm erscheinen (z.B. solche Codes, die eine Bewegung des Cursors auf dem Bildschirm bestimmen), werden ebenfalls im ASCII-Code dargestellt.

Im Anhang B dieses Handbuches wird die gesamte GW-BASIC-Zeichenkette aufgeführt. Die Zeichen, die die Werte 32 bis 126 belegen, sind dem Programmierer bereits bekannt. Einige wenig gebrauchte ASCII-„Control"-Steuerzeichen im Bereich 0 bis 31 verwendet GW-BASIC für graphische Symbole. Obwohl die Werte 128 bis 255 im ASCII-Code nicht vorhanden sind, werden sie bei GW-BASIC verwendet, damit Ihnen eine Vielzahl von zusätzlichen Buchstaben und Symbolen zur Verfügung steht.

Anhand des Anhangs B geben Sie folgende Befehle ein:

```
PRINT CHR$(65)
```

Dieser Befehl fordert GW-BASIC auf, das Zeichen, das dem Codewert 65 entspricht, am Bildschirm anzuzeigen. Der Großbuchstabe A wird somit am Bildschirm ausgegeben (als hätten Sie den Befehl PRINT „A" erteilt). Nun versuchen Sie ein Zeichen, das sich nicht auf der Tastatur befindet, zu schreiben, z.B.

```
PRINT CHR$(227)
```

Als Ergebnis erscheint am Bildschirm der griechische Buchstabe pi. Abschließend versuchen Sie, eines der Steuerzeichen (Zeichen, die nicht am Bildschirm ausgegeben werden) zu schreiben:

```
PRINT CHR$(7)
```

Bei dieser Eingabe ertönt am Lautsprecher ein Warnton.

Die Namen der GW-BASIC Befehle können in Groß- bzw. Kleinbuchstaben eingeben werden. Wenn Sie Ihr Programm das nächste Mal am Bildschirm ausgeben, werden Sie sehen, daß GW-BASIC alle Kleinbuchstaben, die Sie in Befehlen verwendet haben, in Großbuchstaben umgewandelt hat. Dies gilt jedoch nicht für solche Buchstaben, die Sie in Anführungszeichen eingegeben haben: GW-BASIC erkennt, daß Sie beabsichtigen, diese Kleinbuchstaben zu einem späteren Zeitpunkt am Bildschirm bzw. Drucker auszugeben. Wenn Sie zum Beispiel Zeile 0 Ihres Programms wie folgt schreiben:

```
10 print „GROSS klein"
```

wird es bei der nächsten Bildschirmanzeige als

```
10 PRINT „GROSS klein"
```

ausgegeben.

KONSTANTEN

Konstanten sind tatsächliche Daten, die GW-BASIC mitgeteilt werden. Diese Daten können die Form einer Zeichennkette bzw. numerischer Konstanten annehmen. Eine Zeichenkette steht immer in (doppelten) Anführungszeichen z.B.

„WILLKOMMEN"
„Geben Sie jede beliebige Zahl ein"

Dementsprechend enthielt Zeile 20 Ihres Kurzprogramms eine Zeichenkette aus Ihren letzten Übungen (20 PRINT „So kurz kann ein Programm sein").

Sogar Zahlen lassen sich in Anführungszeichen setzen, z.B.

„DM25,000,00"

(in der hier angegebenen Form können jedoch keine arithmetischen Operationen ausgeführt werden).

Wenn Sie mit Zahlen arithmetische Operationen ausführen wollen, können Sie numerische Konstanten für sowohl positive als auch negative Zahlen verwenden. Bei der einfachen Rechenoperation im GW-BASIC-Direktmodus in einer Ihrer letzten Übungen wurden drei numerische Konstanten (128, 64, und 5) verwendet. Es gibt 5 Arten von numerischen Konstanten:

Ganzzahlige Konstanten
: Ganze Zahlen zwischen -32768 und +32767 (bei einer positiven Zahl ist das Pluszeichen wahlfrei).

Festkommakonstanten
: Positive und negative reelle Zahlen, d.h. Zahlen mit Dezimalstellen.

Gleitkommakonstanten
: Konstanten mit positiven bzw. negative Zahlen in Exponentialform (ähnlich der wissenschaftlichen Schreibweise). Eine Gleitkommakonstante besteht aus einer ganzen Zahl bzw. Festpunktzahl (Mantisse) mit wahlfreiem Vorzeichen gefolgt vom Buchstaben E und einer Ganzzahl (dem Exponenten) mit wahlfreiem Vorzeichen. Der zulässige Bereich für Konstanten mit Gleitkomma erstreckt sich von 2.9E-39 bis 1.7E+38.

  Beispiele:
  35E-2 („fünfunddreißig" mal „zehn hoch minus 2")
  = 0.35

235.988E-7 = 0.0000235988
2359E6 = 2359000000

(Gleitkommakonstanten mit doppelter Genauigkeit verwenden den Buchstaben D anstelle von E. Der Unterschied zwischen den beiden wird in diesem Abschnitt beschrieben).

Hexadezimale Konstanten

Zahlen mit &H vorangestellt (Assembler Programmierer wissen darüber gut Bescheid).

Beispiele:
&H76
&H32F

Oktalkonstanten

Zahlen mit &O bzw. &. vorangestellt

Beispiele:
&0347
&1234

Nicht-ganzzahlige numerische Konstanten können Zahlen von einfacher bzw. doppelter Genauigkeit sein. Numerische Konstanten mit einfacher Genauigkeit werden mit bis zu 7 Ziffern gespeichert und mit bis zu 7 Ziffern (mit einer Genauigkeit von 6 Ziffern) am Bildschirm ausgegeben. Bei doppelter Genauigkeit werden die Zahlen mit 17 Ziffern gespeichert und mit bis zu 16 Ziffern am Bildschirm ausgegeben.

Eine Konstante mit einfacher Genauigkeit ist jede nicht-ganzzahlige numerische Konstante, die

- sieben oder weniger Ziffern, bzw.
- Exponentialform mit E , bzw.
- ein Ausrufezeichen (!) aufweist.

Eine Konstante mit doppelter Genauigkeit ist jede numerische Konstante, die

- acht oder mehr Ziffern, bzw.
- Exponentialform mit D, bzw.
- ein nachfolgendes Doppelkreuz (#) aufweist.

Beispiele für Konstanten

Einfache Genauigkeit	Doppelte Genauigkeit
46.8	345692811
-1.09E-06	-1.09432D-06
3489.0	3489.0#
22.5!	7654321.1234

VARIABLEN

Eine Variable stellt eine Art Schublade in Ihrem Programm dar, in der ein Wert (eine Zeichenkette bzw. numerisches Zeichen) aufbewahrt wird. Mit Ihrem Programm können Sie einer Variablen einen Wert zuteilen und ihren Inhalt manipulieren. Sie können sogar die Variablen im Direktmodus mit Werten belegen, sie prüfen und ändern.

Der Name einer Variablen muß mit einem Buchstaben beginnen, der restliche Teil kann sich aus Buchstaben, Ziffern, Dezimalpunkten und einer Typdefinition der Variablen zusammensetzen (siehe unten). Er kann beliebig lang sein. GW-BASIC erkennt jedoch nur die ersten 40 Zeichen an, was kaum eine Beschränkung bedeutet, denn es ist zweckmäßig, den Variablen kurze Namen zu geben, um somit das Programm leichter in den Computer eingeben zu können.

Eine Variable kann entweder eine Zeichenkette oder einen numerischen Wert speichern. Das letzte Zeichen einer Zeichenketten-Variablen muß $ sein und stellt somit die Typendefinition der Variablen dar. Es gibt drei Arten von numerischen Variablen, die jeweils ein Sonderzeichen am Ende des Namens aufweisen:

% Ganzzahl-Variable
! Variable von einfacher Genauigkeit
Variable von doppelter Genauigkeit

Wenn Sie keine Typkennzeichnung festlegen, nimmt GW-BASIC eine Variable von einfacher Genauigkeit an. (Der Unterschied zwischen den verschiedenen Typen der numerischen Werte wurde im Abschnitt „Konstanten") besprochen. Im folgenden finden Sie einige Beispiele für Variablennamen:

P#	kann einen Wert mit doppelter Genauigkeit speichern
MINIMUM!	kann einen Wert von einfacher Genauigkeit speichern
LIMIT%	kann einen Ganzzahlwert speichern
N$	kann einen Zeichenkettenwert speichern

Wenn Sie ein Programm schreiben, können Sie die Namen Ihrer Variablen wählen. Zu den bereits erwähnten Formalitäten der Namensauswahl, kommt noch eine weitere Beschränkung hinzu, der Name einer Variablen darf nicht mit einem zum GW-BASIC-Befehlssatz gehörenden Namen identisch sein. Solche Namen bekommen oft die Bezeichnung „reservierte Wörter", die im Anhang A aufgelistet sind. Ein reserviertes Wort kann jedoch einen Teil eines Variablennamens darstellen. Zum Beispiel sollen Sie in GW-BASIC nicht den Namen PRINT$, sondern die

Bezeichnung PRINTER$ verwenden. Beginnt der Namen einer Variablen mit FN, nimmt GW-BASIC an, daß es ein Aufruf für eine vom Anwender definierte Funktion ist. Dies wird im Kapitel 4 „GW-BASIC Befehle und Funktionen" beschrieben.

Es ist immer zweckmäßig, der Variablen einen Namen zu geben, der in einem sinnvollen Verhältnis zu den in ihr gespeicherten Daten steht. Wenn Sie z.B. die Zinsen für eine Anleihe berechnen wollen, können Sie die laufende Zinsrate in einer Variablen mit der Bezeichnung INTEREST und den Namen des Kreditinstituts mit BANK$ speichern. Alle Kleinbuchstaben, die Sie für einem Variablennamen eingeben, werden von GW-BASIC beim nächsten LIST-Befehl in Großbuchstaben umgewandelt.

Es gibt noch eine zweite Methode der Typkennzeichnung für Variablen, bei der man eine beliebige Anzahl von Variablen mit einem einzigen Befehl (siehe DEFINT, DEFSTR, DEFSNG und DEFDBL im Kapitel „GW-BASIC Befehle und Funktionen") als Ganzzahl-Variable, als Zeichenkette, als Variable von einfacher bzw. doppelter Genauigkeit kennzeichnen kann.

Zur Speicherung von Werten in Variablen steht GW-BASIC der LET-Befehl zur Verfügung. Dazu zwei Beispiele:

```
LET TEMP=30
LET WETTER$=„sonnig"
```

Diese Befehle besagen lediglich, daß die numerische Variable TEMP den Wert 30 annehmen soll, und daß die Zeichenkette WETTER$ die Buchstaben „sonnig" beinhalten sollte. Der Befehl LET wäre eigentlich gar nicht nötig, Sie brauchen nur folgendes einzugeben:

```
TEMP=30
WETTER$
```

Um den jeweiligen Inhalt einer Variablen am Bildschirm ausgeben zu können, bedienen Sie sich des PRINT-Befehls, z.B.:

```
PRINT TEMP:PRINT WETTER$
```

oder vielleicht etwas ausführlicher in der Bedeutung

PRINT „Das Wetter ist ";WETTER$". Die Temperatur beträgt";-TEMP;" Grad."

ANMERKUNG: Wenn einer Variablen nicht ausdrücklich ein Zahlenwert entweder im Direktmodus oder in einem Programm zugewiesen wurde, antwortet es mit 0(numerische Variable) oder gar nicht

(Zeichenketten-Variable), sobald Sie den Inhaltder Variablen zur Ausgabe am Bildschirm aufrufen. Die Länge einer Zeichenketten-Variablen ist die Anzahl der Zeichen, die sich bis zu maximal 255 Zeichen erstreckt.

Wenn Sie versuchen, einer numerischen Variablen einen Zeichenkettenwert zuzuweisen oder umgekehrt, macht Sie GW-BASIC auf den Fehler „Type mismatch" (Typenvermischung) aufmerksam. Was GW-BASIC jedoch nicht kann, ist die Glaubwürdigkeit eines Variableninhalts zu prüfen: wenn Sie z.B. „kalt" dem WETTER$ zuweisen, wird der Satz, der durch den oben erwähnten PRINT-Befehl am Bildschirm ausgegeben wird, etwas widersprüchlich aussehen, obwohl der Befehl von GW-BASIC als syntaktisch richtig angenommen wird.

Es ist möglich, nicht nur Konstanten einer Variablen zuzuordnen, sondern auch die gleichen bzw. andere Variablen zuzuweisen und sogar arithmetische Änderungen bzw. Änderungen an Zeichenketten gleichzeitig vorzunehmen. Wenn Sie TEMP der Zahl 30 und WETTER$ „sonnig" zuordnen, können Sie folgendes ändern:

```
TEMP=TEMP-10
WETTER$=„schön und „+WETTER$
```

Der PRINT-Befehl gibt dann aus:

Es ist ein schöner sonniger Tag und die Temperatur beträgt 20 Grad Celsius.

FELDVARIABLEN

Ein Feld ist eine Gruppe bzw. Tabelle von Werten, die zum gleichen Variablennamen gehören. Jedes Element in einem Feld wird durch den Feldnamen und den Indexwert angesprochen, der GW-BASIC mitteilt, zu welchem Feldelement man Zugang haben will. Beim Index handelt es sich entweder um eine Ganzzahl oder einen Ausdruck, der eine Ganzzahl ergibt. Zum Beispiel teilt PRINT NNAME$(4) GW-BASIC mit, einen Eintrag, der vermutlich aus einer Liste von Namen besteht, am Bildschirm anzuzeigen.

Ein Feld kann mehr als nur eine Dimensionen aufweisen. Ein Beipiel für ein zweidimensionales Feld ist eine Kilometertabelle, welche die Entfernungen zwischen einer Anzahl von Städten angibt. Der Befehl PRINT KM(2,5) weist GW-BASIC an, den numerischen Wert aus der Tabellenposition Spalte 2 und Zeile 5 am Bildschirm anzuzeigen. Die maximale Anzahl der Dimensionen für ein Feld beträgt 255, die maximale Anzahl für jede Dimension 32767.

Der Umgang mit Feld-Variablen unterscheidet sich kaum von dem mit einfachen numerischen Variablen. Soll die Feld-Variable mehr als 11 Elemente (Indices von 0 bis 10) bzw. mehr als eine Dimension enthalten, muß sie dementsprechend benannt werden (siehe DIM im Kapitel „GW-BASIC-Befehle und Funktionen"). Wenn Sie ein Feld verwenden, bevor es definiert wurde, wird angenommen, daß es sich um ein eindimensionales Feld handelt und die höchste Zuweisung 10 aufweist. Dies erfolgte bereits, als ein Wert dem Inhalt eines Elements zugewiesen bzw. der Inhalt gelesen wurde, wobei das gesuchte Element in Klammern angegeben sein muß, wie z.B.:

```
V(2)=65
```

Dabei ist besonders zu beachten, daß hier der Wert 65 dem dritten (!) Element der Feld-Variablen V zugewiesen wird (das erste Element wird mit dem Index 0 versehen, vorausgesetzt Sie ändern dies nicht in 1 mit Hilfe von OPTION BASE).

SPEICHERPLATZBEDARF

Die verschiedenen Typen von Zahlenvariablen benötigen folgenden Speicherplatz:

Ganzzahl einschließlich Hexadezimal- und Oktalzahlbyte – 2 Byte
Variable von einfacher Genauigkeit – 4 Byte
Variable von doppelter Genauigkeit – 8 Byte

Bei einem Feld sind diese Zahlen als Speicherplatz pro Element zu verstehen. Der Speicherplatzbedarf einer Zeichenketten-Variablen ist die Länge der Zeichenkette (Anzahl der Zeichen) plus 3 Byte.

Offensichtlich benötigen die Zahlen mit höherer Genauigkeit mehr Speicherplatz. Sie zu berechnen nimmt mehr Zeit in Anspruch. Ein Programm mit Ganzzahlen läuft daher schneller ab, besonders wenn es sich dabei um immer wiederkehrende Rechenvorgänge handelt.

TYPENUMWANDLUNG

Zuweilen muß für GW-BASIC eine Zahl von einem Genauigkeitsgrad in den anderen umgewandelt werden. Ist dies der Fall, gelten die in diesen Abschnitt aufgeführten Regeln. Wenn Sie versuchen, eine Zeichenketten-Variable einer Zahlenvariablen zuzuweisen oder umgekehrt, meldet sich GW-BASIC mit einem „Type mismatch" (Typenvermischung).

- Wird ein Zahlenwert von einer Genauigkeit einer Zahlenvariablen von unterschiedlicher Genauigkeit zugeordnet, wird die Zahl zusam-

men mit der Genauigkeit, diefür die Zielvariablen (links des Gleichheitszeichens) bestimmt wurde, gespeichert. Beispiel:

```
10 A%= 23.42
20 PRINT A%
```

Wenn Sie dieses kurze Programm ablaufen lassen, gibt GW-BASIC die Zahl 23 aus. Wenn Sie A% die Zahl 23,52 zuordnen, meldet sich GW-BASIC mit der Zahl 24. Die Zahl wird nicht einfach auf eine Ganzzahl gekürzt, sondern abgerundet, sobald der Wert einer Variablen mit niedriger Genauigkeit zugewiesen wird. Eine Abrundung dieser Art findet statt, wenn einer Variablen von einfacher Genauigkeit ein Wert von doppelter Genauigkeit zugeordnet wird.

```
10 C = 55.8834567#
20 PRINT C
```

Am Bildschirm erscheint der Zahlenwert 55.88346.

Diese Abrundung wird dort angewendet, wenn man vergißt, überall dort reine Ganzzahlen in Befehlen bzw. Funktionen zu verwenden, wo Ganzzahlen erforderlich sind. Nehmen wir z.B. an, daß die Variable SUBSC den Wert 2,5 enthält, dann wird beim Programmablauf der Befehl PRINT NNAME$(SUBSCR) von GW-BASIC als PRINT NNAME$(3) interpretiert.

- Weist man einer Variablen von höherer Genauigkeit eine Zahl von niedrigerer Genauigkeit zu, dann ergibt sich natürlich keine größere Genauigkeit. Manchmal erhält man tatsächlich wegen der Art und Weise, wie GW-BASIC Zahlen speichert eine geringfügige Abweichung von der ursprünglichen Zahl. Betrachten wir daher folgendes Beispiel:

```
10 A = 2.04
20 B# = A
30 PRINT A;B#
```

Beim Ablauf dieses Programms gibt GW-BASIC die ursprüngliche und die neue Form der Zahl nebeneinander folgendermaßen aus:

```
2.04 2.039999961853027
```

Für Mathematiker, die eine genaue Information über den Grad der Abweichung benötigen, ist folgender Ausdruck interessant:

```
ABS (B#-A) < 6.3E-8 # A
```

wobei B# und A Variable von doppelter bzw. einfacher Genauigkeit darstellen.

- Bei der Berechnung eines Ausdrucks nehmen alle Operanden Genauigkeitsgrad des genauesten vorhandenen Operanden an. Dazu zwei Beispiele:

```
10 D# = 6#/7
20 PRINT D#
```

Sämtliche arithmetische Berechnungen werden mit doppelter Genauigkeit ausgeführt und auch das Ergebnis (.8571428571428571) wird mit doppelter Genauigkeit ermittelt.

```
10D = 6#/7
20 PRINT D
```

Wieder werden die arithmetischen Berechnungen von doppelter Genauigkeit ausgeführt, diesmal aber wird das Ergebnis einer Variablen von einfacher Genauigkeit zugeordnet. Am Bildschirm erscheint daher beim Befehl PRINT D ein Ergebnis mit einfacher Genauigkeit (0,8571429).

- Logische Operatoren (AND, OR etc, später in diesem Kapitel beschrieben) wandeln ihre Operanden in Ganzzahlen um und man erhält somit ein Ganzzahlergebnis. Die Operanden müssen im Bereich von – 32768 bis 32767 liegen, andernfalls ist ein „Überlauffehler" (Overflow) die Folge.

ÜBUNGEN

Aus dem folgenden Beispiel geht hervor, wie durch die Verwendung von Zeichenketten-Variablen bei der Eingabe Zeit gespart werden kann, wenn die Elemente eines Textes umgestellt werden müssen:

```
10 A$=„The quick brown „:B$=" jumps over the lazy "
20 AN$=„fox":AN2$=„dog":AN3$=„bear"
30 AN4$=„kangaroo":AN6$=„beaver":AN6$=„camel"
40 CLS
50 PRINT A$;AN1$;B$;AN2$:PRINT
60 PRINT A$;AN2$;B$;AN1$:PRINT
70 PRINT A$;AN3$;B$;AN4$:PRINT
80 PRINT A$;AN6$;B$;AN3$:PRINT
90 PRINT A$;AN5$;b$;AN6$:PRINT
100 PRINT A$;AN6$;B$;AN4$:PRINT
```

Jedes Textelement der am Bildschirm ausgedruckten Zeilen ist in einer Zeichenketten-Variablen vorhanden. Die einzelnen PRINT-Elemente sind hier durch Semikolon voneinander getrennt. Das Semikolon weist

GW-BASIC an, das nächste Element unmittelbar im Anschluß an das zuletzt ausgegebene Element am Bildschirm sichtbar zu machen. Der zusätzliche PRINT-Befehl jeweils am Ende der Zeilen 50 bis 100 läßt eine Leerzeile zwischen jeder Zeile am Bildschirm entstehen.

Im nächsten Beipiel wird eine einfache Prozentrechnung an einer Zahlenfolge demonstriert, die dem Computer eingegeben wurde. Das Programm bedient sich dabei der Arithmetik mit einfacher Genauigkeit:

```
 10 PC=5
 20 PCX=PC/100
 30 CLS
 40 INPUT ZAHL
 50 IF NUMBER=0 THEN GOTO 100
 60 ERGEBNIS=ZAHL#PCX
 70 PRINT PC;" % von ";ZAHL" = ";ERGEBNIS
 80 PRINT
 90 GOTO 40
100 END
```

Lassen Sie jetzt dieses Programm ablaufen. Zeile 10 weist PC den Zahlenwert 5 zu. Zeile 20 dividiert diesen Zahlenwert durch 100. Die sich daraus ergebende Zahl kann direkt multipliziert werden, um eine Prozentzahl zu erhalten. Die neue Zahl wird in der Variablen PCX abgelegt. Zeile 30 löscht den Bildschirm. Daraufhin fordert Sie GW-BASIC durch Anzeige eines Fragezeichens auf, einen numerischen Wert (der eine Dezimalstelle enthalten oder als Exponent angegeben sein kann) einzugeben. Der von Ihnen eingegebene Wert wird in ZAHL gespeichert. Zeile 50 prüft, ob ZAHL 0 ist. Ist dies der Fall ist, springt (GOTO) auf Zeile 100, zum Ende (END) des Programms. Ist ZAHL nicht 0, werden 5% von ZAHL berechnet (Zeile 60) und das Ergebnis wird in ERGEBNIS gespeichert. Der Rechenvorgang wird zusammen mit dem Ergebnis am Bildschirm ausgegeben (Zeile 70; Zeile 80 gibt eine Leerzeile aus). Übersteigt das Ergebnis die Anzahl der Ziffern, die am Bildschirm als Zahl von einfacher Genauigkeit angezeigt werden können, bedient sich GW-BASIC automatisch der exponentiellen Darstellung). GW-BASIC springt dann zurück in Zeile 40, was bewirkt, daß GW-BASIC erneut auf die Eingabe einer weiteren Zahl über die Tastatur wartet. Die Befehlsschleife zwischen Zeile 40 und 90 wird solange wiederholt, bis ZAHL den Wert 0 aufweist. Um die Prozentzahl, die Basis für den Rechenvorgang, ändern zu können, ändern Sie einfach dementsprechend Zeile 10. Wenn Sie eine unzulässige Eingabe versuchen, z.B. einen Buchstaben in eine Zahl einbeziehen, fordert Sie GW-BASIC auf, die betreffende Eingabe nochmals vorzunehmen („?Redo from start").

Das letzte Beispiel in diesen Übungen befaßt sich mit dem Gebrauch einer Feldvariablen. Es zeigt uns auch, wie mit einer Mindestzahl von Programmzeilen eine Aufgabe mehrmals wiederholt werden kann, und zwar jedesmal in einer etwas anderen Form. Das Programm bestimmt eine Feldvariable, die aus 10 Elementen in einer einfachen Dimension besteht, und belegt das erste Element mit dem Indexwert 1 (Zeile 10). GW-BASIC fordert Sie nun auf, 10 Namen einzugeben (INPUT), um dieses Feld aufzufüllen (Zeilen 20-50). Daraufhin wird die Anzeige am Bildschirm gelöscht. Sie werden dann aufgefordert, Zahlen nach Ihrer Wahl von 1 bis 10 einzugeben. Nach jeder Eingabe wird das entsprechende Feldelement, das die Namen enthält, am Bildschirm ausgegeben. Bei Eingabe von 0 wird das Programm beendet.

```
 10 OPTION BASE 1:DIM NNAME$(10):CLS
 20 FOR SCHLEIFE%=1 TO 10
 30 PRINT „Geben Sie den Namen von jemandem ein “;
 40 INPUT NNAME$(SCHLEIFE%)
 50 NEXT SCHLEIFE%
 60 CLS
 70 PRINT „Nun rufen Sie diese Namen auf“:PRINT
 80 PRINT „Geben Sie Zahlen von 1 bis 10 ein“:PRINT
 90 INPUT „Zahl“;N%
100 IFN%=0 THEN GOTO 130
110 PRINT „Name “;N%;“ ist “;NNAME$(N%)
120 GOTO 90
130 END
```

Dieses Programm bedient sich einer sogenannten. FOR...NEXT Schleife von Zeile 20 bis 50. Zeile 20 teilt GW-BASIC mit, wie oft die Schleife durchlaufen werden muß (von 1 bis 10, d.h. 10 mal). Die Anzahl der Schleifenabläufe wird in einer Variablen festgehalten, die im Programm mit SCHLEIFE% bezeichnet wird, es kann jedoch jede Zahlenvariable (vorzugsweise ganzzahlig) dafür eintreten. Am Anfang ist die Zahl 1, was bedeutet, daß beim ersten Schleifendurchlauf die Feldvariable NNAME$ in Zeile 40 mit dem Indexwert 1 versehen wird. Dies bedeutet wiederum, daß der erste Name, den Sie bei der Ausführung des Programms eingegeben (INPUT) haben, als erstes Element der Feldvariablen gespeichert wird. Daraufhin erhöht GW-BASIC den in SCHLEIFE% gespeicherten Zahlenwert und prüft, ob der in Zeile 20 festgelegte Grenzwert überschritten wurde. Da dies offensichtlich nach dem ersten Schleifendurchlauf nicht der Fall ist, kehrt GW-BASIC nach Zeile 20 zum nächsten Schleifendurchlauf zurück: zum zweiten Mal werden Sie von GW-BASIC aufgefordert, einen Namen einzugeben; der dann als zweites Element der Feldvariablen gespeichert wird und so wei-

ter. GW-BASIC fährt erst nach dem zehnten Schleifendurchlauf mit Zeile 60 fort.

Zeile 90 gibt am Bildschirm die Aufforderung „Zahl" aus und fordert Sie auf, eine Zahl einzugeben. Die Art wie die Aufforderung in den INPUT-Befehl einbezogen wird, stellt eine Leistung von GW-BASIC dar. Die Eingabe eines ausdrücklichen PRINT-Befehls erübrigt sich. Daher ist Zeile 90 mit

90 PRINT „Zahl";:INPUT N%

gleichwertig. Das Programm hindert Sie jedoch nicht, eine Ganzzahl größer als 10 einzugeben, da sich im Feld kein Element mit dem Indexwert größer als 10 befindet. Wenn Sie jedoch eine solche Zahl eingeben, dann beendet GW-BASIC das Programm mit der Fehlermeldung „Subscript out of range („Indexwert außerhalb des Zahlenbereichs"). Die nächste Reihe von Übungen zeigt Ihnen, wie Sie GW-BASIC programmieren können, um logische Entscheidungen, die solche Vorkommnisse aufdekken, zu treffen und somit verhindern, daß letztere einen vorzeitigen Abbruch des Programms auslösen.

AUSDRÜCKE UND OPERATOREN

Ein Ausdruck kann entweder einfach eine Zeichenkette oder eine Zahlenkonstante bzw. eine Variable sein oder kann Konstanten und Variablen mit Hilfe von Operatoren zu einem einzigen Wert verbinden.

Operatoren führen mathematische bzw. logische Operationen aus, die hauptsächlich an Zahlenwerten vorgenommen werden, dabei besteht jedoch kein Grund dafür, warum Zeichenketten nicht miteinander addiert oder verglichen werden sollten. Die Operatoren von GW-BASIC können so betrachtet werden, als gehörten sie vier verschiedenen Kategorien an:

- Arithmetische
- Vergleichende
- Logische
- Funktionale

ARITHMETISCHE OPERATOREN

Die arithmetischen Operatoren sind folgende:

Operator	Operation	Musterausdrücke
∧	Potenzierung	X∧Y
-	Negation	-X
*,/	Multiplikation, Gleitkomma-Division	X*Y X/Y
+,-	Addition, Subtraktion	X+Y

Diese Operatoren werden hier in einer bestimmten Rangfolge aufgeführt. Wenn Sie ein mathematisches Grundwissen besitzen, ist Ihnen die Notwendigkeit dieser Rangfolge bestimmt bekannt. Sie gibt die Folge wieder, in der untergeordnete Ausdrücke innerhalb eines komplexeren Ausdrucks berechnet werden.

```
2 + 6 # 5
```

Es besteht offensichtlich ein Unterschied im Ergebnis, ob man 2 mit 6 addiert und dann dieses Zwischenergebnis mit 5 (=40) multipliziert oder 5 mit 6 multipliziert und dann 2 addiert (=32). Der übliche Rechenvorgang von beiden ist der letztere, d.h. die Multiplikation hat vor der Addition den Vorrang und wird vor der Addition ausgeführt. Eine solche Rangfolge bezeichnet man als „algebraische Logik".

Bei Verwendung von Klammerausdrücken brauchen Sie diese Rangfolge nicht beachten. Der Ausdruck

```
(2 + 6) * 5
```

ergibt somit 40.

In dem Abschnitt, der sich mit dem GW-BASIC-Zeichensatz befaßt, wurden bereits die Sonderzeichen erwähnt, die mathematische Funktionen darstellen. Ein Beispiel dazu ist das Sternchen (*), das in GW-BASIC den Multiplikationsvorgang bezeichnet. Im folgenden sind einige Beispiele, die zeigen, wie GW-BASIC diese mathematischen Funktionen darstellt:

Algebraischer Ausdruck	GW-BASIC Ausdruck
$X+2Y$	X+Y*2
$X-\frac{Y}{Z}$	X-Y/Z
$\frac{XY}{Z}$	X*Y/Z
$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Z$	(X∧2) ∧Y
X^{Z^Z}	X∧(Y∧Z)
X(-Y)	X*(-Y) Zwei aufeinanderfolgende Operatoren müssen durch eine Klammer voneinander getrennt sein.

Ganzzahl-Division und Modulus-Arithmetik

In GW-BASIC stehen zwei zusätzliche Operatoren zur Verfügung:

Ganzzahl-Division und Modulus-Arithmetik.

Die Ganzzahl-Division wird durch Voranstellen von INT gekennzeichnet. Vor Ausführung der Division werden die Operanden zu Ganzzahlen (müssen im Bereich von -32768 bis 32767 liegen) abgerundet und der Quotient wird auf eine Ganzzahl abgerundet

Zum Beispiel:

10\4 ergibt den Wert 2
25.68\6.99 ergibt den Wert 3

In der Rangfolge kommt die Ganzzahl-Division gleich nach der Multiplikation und Gleitkomma-Division.

Die Modulus-Arithmetik wird durch den Operator MOD gekennzeichnet. Letzterer liefert den Ganzzahlwert, der den Rest einer Ganzzahl-Division darstellt. Überall dort, wo es erforderlich ist, erzeugt GW-BASIC-Ganzzahlen durch Rundung (nicht durch Abrundung). Beispiele:

10.4 MOD 4 = 2(10/4=2 mit dem Rest 2)
25.68 MOD 6.99 = 5(26/7=3 mit dem Rest 5)

In der Rangfolge kommt die Modulus-Arithmetik gleich nach der Division.

Überlauf und Division durch Null

Falls während der Berechnung eines Ausdrucks eine Division durch Null auftaucht, wird die Fehlermeldung „Division durch Null" (DIVISION BY ZERO) angezeigt. Der „Unendlichwert" des Systems mit dem Vorzeichen des Dividenden wird als Ergebnis der Division festgelegt und der Programmablauf wird fortgesetzt. Falls die Berechnung einer Potenzierung eine Null mit negativer Potenz ergibt, wird ebenfalls „Division durch Null" (DIVISION BY ZERO) angezeigt und der positive „Unendlichwert" des Systems wird als Ergebnis der Potenzierung angenommen, bevor das Programm fortgesetzt wird.

Im Falle eines Datenüberlaufs wird die Fehlermeldung „Überlauf" (OVERFLOW) ausgegeben, der Unendlichwert des Systems mit dem algebraisch richtigen Vorzeichen wird als Ergebnis festgelegt und der Programmablauf wird fortgesetzt.

VERGLEICHENDE OPERATOREN

Mit diesen Operatoren werden zwei Werte miteinander verglichen. Das Ergebnis des Vergleichs ist entweder „richtig" (-1) oder „falsch" (0). Es kann dann zur Entscheidung über den weiteren Programmablauf benutzt werden (Siehe IF, Kapitel 4).

Operator	Vergleichstext	Musterausdruck
=	Gleichheit	X=Y
<>	Ungleichheit	X<>Y oder XXY
<	kleiner als	X<Y
>	größer als	X>Y
<= oder =<	kleiner oder gleich	X<=Y oder X=<Y
>= oder =>	größer oder gleich	X>=Y oder X=>Y
(Das Gleichheitszeichen dient auch zur Zuweisung von Werten an Variable. Siehe LET, Kapitel 4).		

Wenn arithmetische und vergleichende Operatoren in einem Ausdruck kombiniert werden , werden immer zuerst die arithmetischen Operationen ausgeführt. Zum Beispiel ist der Ausdruck:

X+Y < (T-1)/Z

„richtig", wenn der Wert von X plus Y kleiner ist als der Wert von T minus 1 geteilt durch Z.

Weitere Beispiele:

```
IF SIN(X) <0,5 THEN GOTO 1000
IF I MOD J<>O THEN K=K+1
```

Im ersten dieser beiden Beipiele bestimmt das Ergebnis der Berechnung (sin X ist kleiner als 0,5), ob der Programmlauf auf Zeile 1000 springt. Im zweiten Beispiel wird der Wert von K um 1 erhöht, vorausgesetzt daß der Rest, der sich aus der Division des Inhalts der Variablen I durch den Inhalt von J ergibt, nicht Null ist. Der Befehl

```
PRINT SECONDS% >30
```

veranlaßt, daß GW-BASIC die Zahl -1 ausgibt, falls die Ganzzahl-Variable einen Wert höher als 30 enthält, andernfalls gibt GW-BASIC 0 am Bildschirm aus.

LOGISCHE OPERATOREN

Logische Operatoren führen logische oder Boolsche Operationen aus. Der Operator definiert die Art des Vergleichs zweier Werte miteinander. Die GW-BASIC-Wörter für die verschiedenen Arten des logischen Vergleichs sind: NOT, AND, OR, XDR, IMP und EQV.

In Ihrem Programm könnte eine Entscheidung bestimmen, ob Sie eine bestimmte Ware aufgrund folgender Kriterien kaufen oder nicht: „Ist der Qualitätscode höher als 3 und liegt der Preis unter 200, dann kaufen Sie!" Ein entsprechender GW-BASIC-Befehl würde dann folgendermaßen aussehen:

```
IF QUALITÄT% > 3 AND PREIS < 200 THEN PRINT
„Die Bedingungen für den Kauf sind gut"
```

Beide Bedingungen müssen erfüllt sein, wenn die Kaufempfehlung am Bildschirm ausgegeben werden soll. Nun betrachten Sie folgendes Beipiel:

```
IF QUALITÄT% > 3 OR PREIS < 200 THEN PRINT
„Die Bedingungen sind für den Kauf annehmbar"
```

was bedeutet, daß wenigstens eine der Bedingungen erfüllt werden muß; und zwar ganz gleich welche; außerdem wäre es von Vorteil, wenn beide Bedingungen erfüllt wären.

Wollen wir nun die gleiche Situation vom Standpunkt des Verkäufers betrachten. Er bzw. sie zieht in Betracht aufgrund folgender Erwägung zu verkaufen: „Ich bin bereit, Ihnen die Ware von einer Qualität von mehr als 3 zu verkaufen, aber dann kann der Preis nicht unter 200 liegen.

„Andernfalls werde ich den niedrigeren Preis akzeptieren, kann jedoch die Qualitätsbedingungen nicht erfüllen." GW-Basic bedient sich des XOR-Operators, um diese Einstellung des „Entweder einer oder der andere, jedoch nicht beide"
auszudrücken z.B. in:

```
IF QUALITÄT% > 3 XOR PREIS < 200 THEN PRINT
„Die Verkaufsbedingungen sind nicht ideal, jedoch gut genug, um
Geschäfte zu machen"
```

Die logischen Operatoren NOT und EQV (gleichwertig mit) haben im GW-BASIC-Zeichensatz jeweils eine Entsprechung.

```
IF NOT (TEMP = 100) THEN GOTO 100) THEN GOTO 1000
hat die gleiche Wirkung wie
IF TEMP <> 100 THEN GOTO 1000

IF ANSWER$ EQV „JA" THEN GOTO 1500
hat die gleiche Wirkung wie
IF ANSWER$ = „JA" THEN GOTO 1500
```

In der folgenden Liste finden Sie das Ergebnis aller Varianten eines jeden der sechs logischen Operatoren. 1 steht für „richtig" und 0 für „falsch". Nimmt man die zweite Variante unter OR als Beispiel, liest sich die Information etwa so „Wenn die erste Bedingung erfüllt ist (1), jedoch die zweite nicht (0), sind die Vergleichsbedingungen insgesamt als erfüllt anzusehen (1). Die erste Variante unter XOR liest sich folgendermaßen: „Wenn sowohl die erste als auch und die zweite Bedingung erfüllt sind, sind die Vergleichsbedingungen insgesamt als nicht erfüllt zu betrachten."

X	NOT X
1	0

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

X	Y	X IMP Y (bedeutet)
1	1	1
1	0	0
0	1	1
0	0	1

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

Diese Liste zeigt ebenfalls die Rangfolge an, in der logische Ausdrücke berechnet werden (NOT als die höchste Priorität). Wie bei arithmetischen Operationen können Sie diese Rangfolge unter Verwendung von Klammern übergehen. Betrachten Sie dazu folgende Beispiele:

```
IF HIMMEL$ = "klar" AND TEMP > 70 OR HUMID < 75
THEN PRINT "Dann wollen wir wandern gehen"

IF HIMMEL$ = "klar" AND (TEMP > 70 OR HUMID < 75)
THEN PRINT "Dann wollen wir wandern gehen"
```

Im ersten Beispiel richtet sich die Einladung zum Wandern nicht nach dem Wetter, sondern nach der Luftfeuchtigkeit, d.h. solange sie unter 75% liegt. Beim zweiten Beispiel müssen sowohl die Temperatur als auch die Luftfeuchtigkeit (bzw. beide) günstig sein und der Himmel muß auf jeden Fall klar sein.

Im folgenden finden Sie eine ausführliche Erläuterung, wie GW-BASIC das Ergebnis einer logischen Operation ermittelt.

Sie brauchen diesen Vorgang nicht verstehen, um mit GW-BASIC programmieren zu können. Die Erläuterung ist hauptsächlich für Programmierer von Interesse, die auf Bit-Ebene arbeiten.

Logische Operatoren wandeln die Operanden in 16-Bit Binärkomplemente zwischen -32768 bis +32767 mit Vorzeichen um. (Wenn die Operanden außerhalb dieses Bereiches liegen, wird ein Fehler angezeigt). Falls beide Operanden 0 oder -1 als Wert ergeben, ermittteln die logi-

schen Operatorenals Ergebnis 0 oder -1 als Wert. Die logische Operation wird mit den umgewandelten Operatoren bit-weise an diesen Ganzzahlen vorgenommen, d.h. jedes Bit aus dem Ergebnis wird durch die entsprechenden Bits der beiden Operanden bestimmt.

Somit ist eine Verwendung logischer Operatoren möglich, um einzelne Byte auf ihr besonderes Bitmuster zu prüfen. Zum Beispiel kann der AND-Operator dazu verwendet werden, um alle Bits eines Statusbytes an einem Maschinen Ein/Ausgang (Port) auszublenden. Der OR-Operator kann verwendet werden, um zwei Byte zu einem bestimmten Binärwert zu mischen. Die nachfolgenden Beispiele sollen die Funktion der logischen Operatoren aufzeigen.

63 AND 16=16	63 = binär 111111 und 16 = binär 10000, daher ist 63 AND 16=16
15 AND 14=14	15 = binär 1111 und 14 = binär 1110, daher ist 15 AND 14 = 14 (binär 1110)
-1 AND 8=8	-1 = binär 1111111111111111 und 8 = binär 1000, daher ist -1 AND 8=8
4 OR 2=6	4 = binär 100 und 2 = binär 10, daher ist 4 OR 2 = 6 (binär 110)
1O OR 1O=10	10 = binär 1010, daher ist 1010 OR 1010 = 1010(10)

Sie können GW-BASIC dazu verwenden, das Zweierkomplement einer Ganzzahl zu errechnen:

ZWEIKOMP% = (NOT GANZZAHL%) + 1

Beispiel: GANZZAHL% enthält den Wert 2 (=binär 10), NOT GANZZAHL% erzeugt das Bitmuster 1111111111111101. Dezimal ausgedrückt bedeutet dies -3. Dem ZWEIKOMP% wird daher der Wert -2 zugeordnet (das Ergebnis aus der Addition von 1 + -3). Der allgemeine Ausdruck zur Berechnung des Zweierkomplements einer Ganzzahl ist „Bitkomplement plus 1".

FUNKTIONALE OPERATOREN

Funktionen werden in einem Ausdruck verwendet, um eine vorbestimmte Operation für einen Operanden aufzurufen. GW-BASIC besitzt interne Funktionen, die bereits im System integriert sind, wie z.B. SQR (Quadratwurzel) oder SIN (Sinus). Alle internen Funktionen von GW-BASIC sind in Kapitel 4 beschrieben.

GW-BASIC ermöglicht auch dem Programmierer, selbst Funktionen zu definieren. Siehe DEF FN, Kapitel 4.

BERECHNUNG VON AUSDRÜCKEN

Dieser Abschnitt faßt die Rangfolge der numerischen Operationen zusammen, d.h. die Reihenfolge, in der GW-BASIC solche Operationen in einem Ausdruck ausführt.

1. Funktionsaufrufe (ungeachtet ob in Ihrem Programm definiert oder bereits von GW-BASIC bereitgestellt) werden zuerst ausgerechnet.

2. Arithmetische Operationen werden dann in folgender Reihenfolge ausgeführt:

   a ∧
   b monadisch –
   c. *,/
   d. \
   e.MOD
   f. +,–

3. Vergleichende Operationen werden als nächstes ausgeführt.
4. Zuletzt werden logische Operationen in der folgenden Reihenfolge ausgeführt.

   a. NOT
   b. AND
   c. OR
   d. XOR (exclusive OR = ausschließlich)
   e. EQV (equivalent = gleichwertig)
   f. IMP (implication = Folgerung)

Von links nach rechts werden Operationen in der gleichen Reihenfolge ausgeführt. Um die Rangfolge für einen besonderen Ausdruck zu ändern, sind Klammern zu benutzen: die Operationen in Klammern erfolgen zuerst und die übliche Priorität innerhalb der Berechnungen (wie oben genau beschrieben) wird eingehalten.

OPERATIONEN MIT ZEICHENKETTEN

Zeichenketten werden mit einem Pluszeichen (+) verkettet, z.B.:

```
10 A$="DATEI":B$="NAME"
20 PRINT A$ + B$
30 PRINT"NEUER" + A$ + B$
```

RUN
DATEINAME
NEUER DATEINAME

Zeichenketten können mit den gleichen Vergleichsoperatoren, die bei Zahlen verwendet werden, verglichen werden.

= <> >< <= >=

Vergleiche mit Zeichenketten werden so ausgeführt, daß die betreffenden Zeichenketten unter Berücksichtigung des ASCI-Wertes Zeichen für Zeichen verglichen werden. Wenn alle ASCI-Werte gleich sind, sind auch die Zeichenketten gleich. Sind die ASCI-Werte verschieden, kommen die niedrigeren Codezahlen vor den höheren. Wenn im Verlauf einer Vergleichsoperation das Ende einer Zeichenkette erreicht ist, dann wird die kürzere Zeichenkette als die kleinere angesehen. Vorangestellte und nachfolgende Leerstellen sind von Bedeutung. Beispiele (das Ergebnis ist in jedem Fall „richtig"):

"AA2" <"AB"
"DATEINAME" = "DATEINAME"
"X&" > "x#"
"CL" > "CL"
"kg" > "KG"
"SCHMIDT" < "SCHMITT"
B$ < "256" (wobei B$ die Zeichenkette "1234" enthält)

Vergleichsoperationen mit Zeichenketten können zur Prüfung bzw. zur Alphabetisierung von Zeichenketten verwendet werden. Alle in Vergleichsausdrücken verwendete Zeichenkettenkonstanten müssen in Anführungsstrichen stehen.

ÜBUNGEN

Sie haben jetzt eine ganze Menge über die Priorität von Operatoren gelesen. Machen Sie jetzt den Versuch, einige mathematische Operationen einzugeben und dabei besonders auf die Auswirkungen zu achten, die sich ergeben, wenn mit Klammerausdrücken gearbeitet wird und so die übliche Rangfolge übergangen werden. Achten Sie z.B. auf den Unterschied bei dem Ergebnis zwischen

PRINT 5 + 6 * 12

und

PRINT (5 + 6) * 12

Geben Sie diese Befehle im Direktmodus ein, damit Sie ein schnelles Ergebnis erhalten. Sie können sogar GW-BASIC-Variable in einen Ausdruck einbeziehen, was jedoch nur dann einen Sinn hat, wenn Sie ihnen zuerst einen Zahlenwert zuordnen, andernfalls weist GW-Basic diesen Variablen den Wert 0 zu. Geben Sie jetzt im Direktmodus ein:

```
X = 12
Y = 14
```

und darauffolgend einen Befehl (ebenso im Direktmodus), um einen Ausdruck, der zwei Variable verwendet, aufzulösen, wie zum Beispiel:

```
PRINT Y MOD X + 3
```

Das nächste Beispiel wird Sie an die arithmetischen Rechenübungen Ihrer Schulzeit erinnern, bevor Sie etwas über Dezimalstellen gelernt haben. Es zeigt Ihnen, wie Sie bei GW-BASIC ganzzahlige und Modulus-Divisionsoperatoren verwenden können, um ein Ergebnis in Form eines Quotienten und Restbetrages zuerhalten.

Das Programm bedient sich der sogenannten „Fehlerbehandlung". Sobald GW-BASIC beim Programmablauf bemerkt, daß ein synktatischer Fehler vorliegt, oder daß ein Befehl bzw. eine Funktion nicht richtig verwendet wurde (z.B. Division durch Null oder nichtzulässiger Index), meldet sich dazu GW-BASIC am Bildschirm. Sie können nicht-syntaktische Fehler „abfangen", indem Sie den Inhalt einer Variablen befragen und je nach Antwort im Programm einen Sprung ausführen. Bei den letzten Übungen, in dem Programm, das eine Liste von Namen gespeichert hat und das sich dann aufgrund Ihrer Zahleingabe an diese Namen erinnerte, wurden Sie von nichts daran gehindert, eine Zahl höher als der maximal zulässige Index für das Feld einzugeben.

Ein Weg, um eine solche fehlerhafte Eingabe „abfangen" zu können, wäre folgende Programmzeile:

```
105 IF N% > 10 THEN PRINT "So viele Namen gibt es nicht":
    GOTO 80
```

Dies bewirkt, daß, wenn Sie eine Zahl höher als 10 eingeben, zunächst Ihre eigene Fehlermeldung am Bildschirm erscheint, worauf GW-BASIC nach Zeile 80 zurückkehrt und auf eine zulässige Eingabe wartet. So verhindert Ihr Programm, daß GW-BASIC versucht, in Zeile 110 auf ein Feld mit einem zu hohen Index zu verweisen.

GW-BASIC besitzt eine noch bessere Möglichkeit zur Fehlerbehandlung, die im folgenden Programm angewendet wird. Zeile 10 fordert GW-BASIC auf, im Fall eines Fehlers, der irgendwo im Programm auftauchen

kann, auf Zeile 100 zu springen. Die Fehlerarten, die in diesem Programm auftreten können, würden sich aus der Eingabe einer Ganzzahl ergeben, die außerhalb des zulässigen Ganzzahl-Bereichs liegt (aufgrund eines Überlauffehlers) oder aus dem Versuch, GW-BASIC zu veranlassen, eine Division mit Null vorzunehmen (Fehlermeldung bei Division mit Null). Zeile 110 befaßt sich mit dem früheren Fehler, mit diesem Fehler braucht man sich jedoch nicht abgeben, da eine Division durch Null GW-BASIC nicht veranlaßt, den Programmablauf zu stoppen. Die Zahlen 6 und 11 sind Codes, deren sich GW-BASIC bedient, um diese Fehler zu kennzeichnen. Im Anhang C dieses Handbuchs finden Sie eine Liste sämtlicher Fehlermöglichkeiten, die GW-BASIC erkennen kann. Es ist besonders wichtig, den Fehler bei Ganzzahlüberlauf einzugrenzen, da ein solcher Fehler sonst zur Beendigung des Programms führen könnte. Der RESUME 30 Befehl fordert GW-BASIC auf, den Fehler zu berücksichtigen und den normalen Programmablauf in Zeile 30 wieder aufzunehmen.

Sie werden schon bemerkt haben, daß es bei diesem Programm nicht möglich ist, die Befehlsschleife zwischen Zeile 30 und 70 zu verlassen. Dies verschafft Ihnen die Gelegenheit, die Steuertaste BREAK zu benutzen. Sie beendet ein Programm und bringt GW-BASIC wieder auf seine Bereitschaftsebene („Ok") zurück. Bei der Entwicklung von Programmen werden Sie wahrscheinlich von dieser Möglichkeit häufig Gebrauch machen, um eine unendliche Programmschleife verlassen zu können.

```
 10 ON ERROR GOTO 100
 30 INPUT "zu teilende Zahl";Q%
 40 INPUT "Jetzt geben Sie den Divisor ein";D%
 50 PRINT:PRINT "Die Antwort darauf ist ";Q%\!D%;", Rest ";Q%
    MOD D%
 60 PRINT
 70 GOTO 30
100 CLS
110 IF ERR=6 THEN PRINT "Außerhalb des Ganzzahlbereiches!
    Noch einmal versuchen"
130 RESUME 30
```

Der Abschnitt „Zeichenketten-Operatoren" hat gezeigt, daß es möglich ist, nicht nur Zahlen, sondern auch Zeichenketten miteinander zu vergleichen. Das Ergebnis aus dem Vergleich zweier Zeichenketten hängt davon ab, wie Zeichen für Zeichen mit dem ASCII-Wert verglichen werden. Somit ist „345" als Zeichenkette gesehen größer als „12345", weil der ASCII Code für die Ziffer „3" höher ist als die für die Ziffer „1".

Das folgende Programm fordert Sie auf, 10 Namen einzugeben (Zeile 10 bis 50), die im Feld NNAME$ gespeichert werden. Am Bildschirm

erscheint keine Anzeige und das Programm läuft weiter, um die Namen in aufsteigender ASCII-Folge (Zeilen 70 bis 110) zu sortieren. Es gibt viele bekannte Verfahren zur Datensortierung. Das vorliegende Verfahren ist jedoch eines der einfachsten. Bei diesem Verfahren wird die Liste durchgegangen und eine Reihe von benachbarten Einträge werden miteinander verglichen, und wo erforderlich erfolgt ein Datenaustausch. In Zeile 110 prüft das Programm die Variable TAUSCH$ auf „Y" für Ja oder auf „N" für nein, um zu sehen, ob ein Datenaustausch beim gerade erfolgten Durchgang der Liste erforderlich war. War dies der Fall, wird die Liste noch einmal durchlaufen. Nach einem Leer-Durchlauf, bei dem sich ein Datenaustausch erübrigt, läuft das GW-BASIC Programm weiter, um die Liste in der neuen Reihenfolge auszudrucken.

Beachten Sie jedoch, daß der Sortiervorgang den gesamten ASCII-Code berücksichtigt und sich nicht nur auf Buchstaben beschränkt. Das Programm hindert Sie nicht daran, nicht-alphabetische Zeichen einzugeben. Darüber hinaus können Sie bestimmen, ob Groß- oder Kleinbuchstaben verwendet werden sollen. Sie könnten z.B. den „Namen" "!seltsam" eingeben. Der Sortiervorgang würde dann dieses Wort näher an den Anfang der Liste setzen wie jeden anderen Namen, der mit einem Buchstaben beginnt, weil der ASCII-Wert für das Ausrufezeichen 57 beträgt und der niedrigste Code, der mit einem Buchstaben belegt ist, 65 (großes A) ist.

Der tatsächliche Datenaustausch erfolgt in einem „Unterprogramm" in Zeile 180. Dieses Unterprogramm wird mit dem GOSUB-Befehl in Zeile 90 (falls die Bedingung zum Datenaustausch erfüllt ist) eingegeben. Der RETURN-Befehl fordert GW-BASIC auf, zu den Befehl zurückzukehren, der unmittelbar auf dem folgte, der GW-BASIC in das Unterprogramm schickte, In unserem Fall nach Zeile 100. Hier wäre es möglich gewesen, einen Befehl GOTO 180 zu benutzen, um den Namen des Unterprogramms für den Datenaustausch und den Befehl GOTO 100 eingeben zu können, damit GW-BASIC wieder zurückfinden kann. Der Unterschied zwischen den beiden Methoden liegt darin, daß der RETURN-Befehl (kann nur zusammen mit GOSUB verwendet werden) GW-BASIC wieder an den richtigen Ort zurückbringt, ohne eine Zeilennummer nennen zu müssen. Dies ist eine sehr zweckmäßig in solchenb Programmen, bei denen ein Unterprogramm an verschiedenen Stellen eingegeben werden kann.

Der größere Teil der „Tausch"-Routine" ist damit beschäftigt, die Namenpaare, die ausgetauscht werden am Bildschirm auszugeben. Die Programmschleife in Zeile 200 stellt für den Programmablauf eine Verzögerung dar, denn das Programm muß diese Schleife 600mal durchlaufen, bevor es zurückkehren (RETURN) kann, um das nächste Namenpaar vergleichen zu können. Die Schleife selbst enthält keinerlei

Befehle, so bewegt sich hier GW-BASIC tatsächlich am Fleck, um Ihnen genügend Zeit zu geben, die Anzeige am Bildschirm lesen zu können, bevor das nächste Namenspaar aus dem Unterprogramm am Bildschirm erscheint. Ist die Ablesezeit zu kurz, ist die Anzahl der Schleifendurchgänge zu erhöhen. Sie sollten jedoch dabei beachten, daß die Anzeige der aus dem Unterprogramm gefundenen Namenspaare die Zeit beträchtlich erhöht, die GW-BASIC für den gesamten Sortiervorgang benötigt. Um die Sortiergeschwindigkeit zu erhöhen, sind die Zeilen 180 und 200 zu löschen.

Zeile 170 enthält keine Befehlef für GW-BASIC, stellt jedoch einen Kommentar des Programmierers dar. Trifft GW-BASIC auf das Wort REM, braucht es den Rest der Zeile nicht mehr zu lesen, daher können Sie in diese Zeile eingeben, was Sie wünschen. In unserem Fall wird sie für zwei Zwecke verwendet: die Sternchen machen eine klare Trennung zwischen dem Ablauf des Unterprogramms und des Hauptprogramms, und erleichtert somit dem Programmierer das Lesen des Programms. Nachfolgende Anmerkung bezieht sich auf den Zweck des Unterprogramms

```
 10 OPTION BASE 1:DIM NNAME$(10):CLS
 20 FOR SCHLEIFE%=1 TO 10
 30 PRINT "Geben Sie den Namen von jemandem ein";
 40 INPUT NNAMES$(LOOP%)
 50 NEXT SCHLEIFE%
 60 CLS
 70 TAUSCH$="N"
 80 FOR LP%=1 TO 9
 90 IF NNAME$(LP%)>NNAME$(LP%+1) THEN TAUSCH$=
    "Y":GOSUB 180
100 NEXT LP%
110 IF TAUSCH$="Y" THEN CLS:GOTO 70
120 PRINT "InASCII-Reihenfolge":PRINT
130 FOR LP%=1 TO 10
140 PRINT NNAMNE$(LP%)
150 NEXT LP%
160 END
170  REM ********** Unterprogramm - Feldelemente
    anzeigen/austauschen
180     PRINT "Tauschen von";NNAME$(LP%);" und
    ";NNAME$(LP%+1)
190 SWAP NNAME$(LP%),NNAME$$(LP%+1)
200 FOR DLY%=1 TO 800:NEXT DLY%
210 RETURN
```

Bildschirmeditor

Dieser Abschnitt beschreibt die Verwendung der Tastatur Ihres Computers in Hinblick auf das Schreiben und Editieren von GW-BASIC-Programmen. Wenn Sie mit der Belegung und den Grundfunktionen der Tastatur noch nicht vertraut sind, sollten Sie zunächst die entsprechende Beschreibung in der Betriebsanleitung Ihres Computers lesen.

GW-BASIC hat einen komfortablen Bildschirmeditor, mit dem Sie Programme erstellen und ändern können. Das Editieren beschränkt sich nicht auf die augenblicklich geschriebene Programmzeile, sondern Sie können sich der einzelnen Cursortasten bedienen, um den Cursor an jede beliebige Stelle des Bildschirms zu bewegen, um an dem vom Cursor angezeigten Punkt Zeichen löschen bzw. einfügen zu können. Nachdem Sie eine Programmzeile editiert haben, betätigen Sie die Abschlußtaste. Während sich der Cursor noch in der betreffenden Zeile befindet, speichert GW-basic den neuen Inhalt der betreffenden Zeile.

ANMERKUNG: Der GW-BASIC-Befehl NEW löscht den Speicherinhalt jedes GW-BASIC-Programms, das sich augenblicklich im Speicher befindet, läßt jedoch GW-BASIC unberührt. Der NEW-Befehl wird dazu verwendet, um sich zu vergewissern, daß der Inhalt des Speichers gelöscht wurde, bevor man mit dem Editieren eines neuen Programms beginnen kann.

Der Cursor wird durch Betätigen einer einzigen Taste bewegt. Die Bewegungsrichtung ist auf jeder der vier betreffenden Tasten gekennzeichnet. Wenn sich der Cursor zu weit nach links über den Rand des Bildschirms hinaus bewegt, erscheint er sofort wieder am äußersten rechten Rand der vorausgehenden Zeile. Bewegt sich der Cursor über den rechten Rand des Bildschirms, erscheint er wieder am äußersten linken Rand der nächsten Zeile.

Die einzelnen Tastaturfunktionen, die von GW-BASIC unterstützt werden, werden nachfolgend aufgeführt. Bei einigen muß eine Taste in Verbindung mit der Control-Taste betätigt werden.

Taste	Funktion
<Home>	Der Cursor bewegt sich zur linken Ecke des Bildschirms
<Control-Home>	Die Anzeige am Bildschirm wird gelöscht und der Cursor bewegt sich in die oberste linke Ecke des Bildschirms
↑	Der Cursor bewegt sich eine Zeile nach oben.
↓	Der Cursor bewegt sich eine Zeile nach unten.
←	Der Cursor bewegt sich nach links
→	Der Cursor bewegt sich nach rechts
<Control-→>	Der Cursor bewegt sich nach rechts bis zum Anfang des nächsten Wortes. Der Anfang eines Wortes ist der erste Buchstabe bzw. Ziffer, die auf eine Leerstelle bzw. Sonderzeichen folgt (z.B. Interpunktionszeichen).
<Control-←>	Der Cursor bewegt sich nach links bis zum Anfang des vorausgehenden Wortes.
<End>	Der Cursor bewegt sich an das Ende der logischen Zeile, d.h. an das Ende der GW-BASIC-Zeile (die über den Rand einer Bildschirmzeile hinausgehen kann). Diese Funktion wird besonders dann benützt, wenn eine schon bestehende Programmzeile erweitert werden soll.
<Control-End>	Die logische Zeile der augenblicklichen Cursor-Position bis zum Ende der Zeile wird gelöscht.
<Ins>	Diese Taste dient zum Ein-/ Ausschalten beim Einfügemodus. Ist der Einfügemodus ausgeschaltet und wird die Taste <Ins> gedrückt, wird der Einfügemodus ein bzw. ausgeschaltet Ist der Einfügemodus eingeschaltet, verschieben die eingegebenen Zeichen die vorhandenen nach rechts. Wenn in der gleichen Bildschirm-

zeile kein Platz mehr ist, werden die Zeichen in die nächste Zeile befördert. Dabei gehen keine Zeichen verloren, weder in der Cursor-Zeile noch in den darauffolgenden Zeilen. Solange der Einfügemodus eingeschaltet ist, bedeckt der Cursor die untere Hälfte der Zeichenposition.

Solange der Einfügemodus ausgeschaltet ist, werden die vorhandenen Zeichen durch die augenblickliche Eingabe überschrieben (ersetzt). Zu dem obengenannten Ein-/Ausschalteffekt kommt noch hinzu, daß beim Betätigen der Cursor- bzw. der Abschlußtaste der Einfügemodus ausgeschaltet wird.

<Del> Das Zeichen, an dem sich der Cursor gerade befindet, wird gelöscht. Zeichen rechts vom Cursor bewegen sich nach links, um die Leerstelle zu beseitigen. Der Vorgang, bei dem die Leerstelle beseitigt wird, bezieht sich auf die gesamte logische Zeile, die auf die augenblickliche Cursorposition folgt.

|← Rücktaste. Mit dieser Taste wird jedes Zeichen unmittelbar links vom Cursor gelöscht. Die so entstandene Leerstelle kann durch die Löschfunktion (DELETE) beseitigt werden.

<Esc> Bei Betätigung dieser Taste wird die gesamte logische Zeile, in der sich der Cursor augenblicklich befindet, am Bildschirm gelöscht. Wenn jedoch die Zeile bereits an GW-BASIC abgeschickt wurde (Betätigung der Abschlußtaste), wird sie nicht aus dem Programm gelöscht. (Um eine Zeile aus dem Programm zu löschen, ist einfach die Zeilennummer einzugeben und die Abschlußtaste (CR) zu betätigen bzw. der DELETE-Befehl zu benutzen.)

<Control-Break> GW-BASIC kehrt auf die Befehlsebene („Ok") zurück, ohne irgendwelche Veränderungen, die an der gerade editierten Zeile vorgenommen wurden, zu speichern. Die Zeile bleibt am Bildschirm erhalten. (Bei Betätigung der Abschlußtaste (CR) wird an GW-BASIC eine neue Zeile übergeben).

<Tab> Wenn der Einfügemodus ausgeschaltet ist, kann man mit dieser Taste den Cursor zur nächsten Tabulatorposition bewegen, ohne irgendwelche Zeichen zu verschieben. Die Tabulatorpositionen sind in einer Zeile auf jeweils acht Zeichenstellen verteilt, d.h. an den Positionen 9,17,25 etc.

Wenn der Einfügemodus eingeschaltet ist, werden Leerstellen von der augenblicklichen Cursorposition bis zur nächsten Tabulatorposition eingefügt. Sobald eingefügt wird, wird der Text verschoben (siehe oben).

<CR> Eingabetaste (manchmal Abschlußtaste genannt). Bei Betätigung dieser Taste wird die Programmzeile, in der sich der Cursor augenblicklich befindet, an GW-BASIC übergeben. Bis zur nächsten Änderungen (wie es GW-BASIC bei Betätigung der Abschlußtaste mitgeteilt wurde) ist dies die Zeile, wie sie von GW-BASIC beim Programmablauf gesehen wird.

<Control<CR>> Die Betätigung dieser Taste bewirkt einen sogenannten Zeilenvorschub, d.h. der Cursor springt zur nächsten Zeile, jedoch die Programmzeile wurde noch nicht an GW-BASIC übergeben. Die so entstandenen Leerstellen am Ende der oberen Zeile haben keinerlei Auswirkung auf den Inhalt des Programms. Diese Funktion ist besonders dann zu verwenden, wenn man die Zeilen aufteilen will, um ein Programm besser lesen zu können.

<Control-PrtSc> Diese Tastenkombination entspricht nicht der normalen Shift-Prtc Funktion. Control-PrtSc weist den Drucker an, alles, was auf dem Bildschirm erscheint, auszudrucken, und zwar nicht nur beim Editieren, sondern auch während des Programmablaufs. Diese Funktion bleibt solange bestehen, bis die Control-PrtSc-Taste erneut betätigt wird.

Die wirksamste Methode, die verschiedenen Editorfunktionen zu lernen, ist die Übung. Laden Sie einfach GW-BASIC wie üblich und schreiben Sie einige Programmzeilen. Beachten Sie jedoch dabei, daß jede Programmzeile aus einer Zeilennummer gefolgt von mindestens einer Leerstelle besteht, an die sich der Programmtext anschließt. Die Zeilen, die sie eingeben, während Sie am Bildschirmeditor üben, brauchen keineswegs nach perfekter GW-BASIC-Syntax geschrieben sein. Sie werden wahrscheinlich keinen Programmablauf versuchen wollen. Sie könnten dazu die Beispiele aus den Übungen von Kapitel 1 „Einbauen von Fehlern und deren Berichtigung" verwenden.

Hier sind einige Vorschläge, wie man Programme besser schreiben und editieren kann:

- Beachten Sie, daß nur bei Betätigung der Abschlußtaste eine Programmzeile von GW-BASIC zur Kenntnis genommen wird. Ganz gleich in welcher Zeile sich der Cursor befindet, wenn Sie die Abschlußtaste betätigen, nimmt GW-BASIC die gesamte Zeile zur Kenntnis. Sie brauchen dazu den Cursor nicht bis zum Zeilenende bewegen.

- Um eine Zeile löschen zu können, brauchen Sie nur die Zeilenzahl eingeben oder DELETE verwenden. Der DELETE-Befehl dient zum Löschen einer Anzahl fortlaufender Zeilen.

- Mit der Eingabe des LIST-Befehls können Sie einen Teil bzw. Ihr gesamtes Programm einsehen. Wo immer passend (z.B. Wörter, die für GW-BASIC reserviert sind), verwendet LIST Großbuchstaben und läßt die Programmzeile in aufsteigender numerischer Reihenfolge am Bildschirm erscheinen. Wenn Sie eine größere Anzahl von Zeilen auf einmal einsehen wollen, werden Sie bemerken, daß der am Bildschirm dargestellte Text zu schnell nach oben rollt, um überhaupt gelesen werden zu können. Die Betätigung der Tastenkombination Control-Num Lck hebt das Rollen am Bildschirm für kurze Zeit auf und gibt Ihnen genügend Zeit, um Ihren Text in Ruhe lesen zu können. Um den Text erneut rollen zu lassen, betätigen Sie einfach jede beliebige Zeichentaste (die gleiche Tastenkombination unterbricht auch den Programmablauf, wobei Sie am Bildschirm lange Listen bzw. andere von einem Programm erstellte Texte lesen können).

- Manchmal erfordert ein Programm eine Anzahl von Zeilen mit beinahe gleichem Inhalt. In diesem Fall ist es äußerst zweckmäßig, eine Zeile zu doppeln, um dann die erforderlich kleinen Änderungen in der doppelten Zeile vornehmen zu können.

Zur Verdopplung einer Zeile ist der Cursor an den Anfang der zu verdoppelnden Zeile zu setzen. Dann überschreiben Sie die Zeilennummer mit der neuen Zeilennummer für die kopierte Zeile. Betätigen Sie nun die Abschlußtaste, daraufhin erkennt GW-BASIC die kopierte Zeile. Die ursprüngliche Zeile wird davon nicht betroffen.

- Um eine Zeilennummer zu ändern, kopieren Sie die Zeile, wie oben beschrieben und löschen Sie dann die ursprüngliche Zeile.

- Der AUTO-Befehl liefert Ihnen die Zeilennummern und erspart Ihnen somit, diese einzeln eingeben zu müssen. Bevor Sie Zeilen editieren, die sich von der augenblicklich bearbeiteten Zeile unterscheiden, sollten Sie die AUTO-Funktion durch Betätigung von Control-Break abschalten.

- Eine Anzahl von GW-BASIC reservierten Wörtern kann mit Hilfe der Alt-Tastenkombination eingegeben werden, so läßt z.B. die Tastenkombination Alt-P das Wort PRINT am Bildschirm erscheinen. Im folgenden finden Sie die komplette Liste für diese speziellen Alt-Tastenfunktionen:

A AUTO	N NEXT
B BSAVE	O OPEN
C COLOR	P PRINT
D DELETE	Q nicht verwendet
E ELSE	R RUN
F FOR	S SCREEN
G GOTO	T THEN
H HEX$	U USING
I INPUT	V VAL
J nicht verwendet	W WIDTH
K KEY	X XOR
L LOCATE	Y nicht verwendet
M MID $	Z nicht verwendet

- Zehn der Funktionstasten auf der Tastatur werden mit den für GW-BASIC reservierten Wörtern programmiert, sobald GW-BASIC in den Arbeitsspeicher geladen wird. Hierbei handelt es sich um Befehle, die besonders im Direktmodus anwendbar sind, und somit nach Betätigung der Abschlußtaste zur Verfügung stehen. Um einen dieser Befehle zu aktivieren, ist die entsprechende Funktionstaste zu betätigen:

F1 LIST	F2 RUN <CR>
F3 LOAD"	F4 SAVE"
F5 CONT <CR>	F6 ,"LPT:" <CR>
F7 TRON <CR>	F8 TROFF <CR>
F9 KEY	F10 SCREEN 0,0,0 <CR>

GW-BASIC verwendet die unterste Zeile des Bildschirms, um das Inhaltsverzeichnis dieser Funktionstasten darzustellen. Dieser Teil des Bildschirm kann ein- und ausgeschalten werden ebenso kann die Belegung dieser Tasten mit Hilfe des KEY-Befehls geändert werden.

- Wenn Sie auf einem Teil des Bildschirms wichtige Informationen stehen lassen und auf dem anderen Ihr Programm editieren wollen, sollten Sie die Beschreibung des VIEW-Befehls in Kaptel 4 lesen.

- Programmänderungen finden im Arbeitsspeicher statt. Sie werden erst bei Erteilung des SAVE-Befehls auf Platte gespeichert. Wenn Sie ein langes Programm schreiben, sollten Sie mehrmals SAVE-Befehle eingeben. In der Regel werden Sie jedes Mal den gleichen Dateinamen verwenden, damit nicht frühere weniger vollständige Versionen Ihres Programms Ihre Platte anfüllen. Ein Programm, das einmal auf Platte gespeichert ist, kann nicht einmal durch einen Stromausfall gelöscht werden.

Anzeige am Bildschirm

GW-BASIC bietet die Möglichkeit, sowohl Text (einschließlich Sonderzeichen, wie im Kapitel 1 „Der Zeichensatz" beschrieben) als auch Punkte und geometrische Figuren am Bildschirm auszugeben. Bei einfarbiger Bildschirmanzeige können Sie alle Zeichen am Bildschirm darstellen einschließlich der graphischen Zeichen (siehe Anhang B) des GW-BASIC Zeichensatzes. Sie können auch folgende Bildschirmeigenschaften beeinflussen: Kontrast, Bildinversion (dunkle Schrift auf hellem Hintergrund und umgekehrt), Unterstreichung und Blinken. Bei einem Farbbildschirm haben Sie die zusätzliche Möglichkeit der graphischen Zeichnung mit zwei verschiedenen Auflösungen, außerdem besitzt der Bildschirm einen 16-KByte-Buffer, der 4 Bildschirmseiten mit einer Breite von 80 Zeichen bzw. 8 Seiten mit einer Breite von 40 Zeichen speichern kann.

TEXTMODUS

Im Textmodus geht GW-BASIC davon aus, daß der Bildschirm 25 Zeilen aufweist, die mit dem Text des GW-BASIC-Zeichensatzes, beschrieben werden können. Jede Zeile umfaßt 40 oder 80 Zeichen (die Zeilenlänge wird mit dem WIDTH-Befehl gesetzt).

Bei GW-BASIC wird die oberste linke Zeichenposition als Zeile 1, Spalte 1 betrachtet; die Zeichenposition rechts unten am Bildschirm ist 25;80 (wenn angenommen wird, daß Ihr Programm 80 Spalten pro Zeile ausgewählt hat). Die 25. Zeile ist für ein GW-BASIC Programm zugänglich, läßt sich jedoch nicht am Bildschirm rollen. Diese Zeile zeigt normalerweise die augenblickliche Belegungen der Funktionstasten auf Ihrer Tastatur an.

Wie ein einzelnes Zeichenfeld dargestellt wird, richtet sich nach der Vorder- und Hintergrundfarbe, wobei der Vordergrund das Zeichen selbst und der Hintergrund das kleine, rechteckige das Zeichen umgebende Feld darstellt. Auf einem Farbgerät lassen sich sowohl die Vordergrund- als auch die Hintergrundfarben in einem GW-BASIC-Programm setzen. Ein Zeichenfeld blinkend darzustellen ist ebenso möglich. Bei einem

Farbgerät stehen 16 Farben zur Verfügung (die Zahl bei jeder Farbe bedeutet die GW-BASIC Kennummer).

Die im Textmodus zur Verfügung stehenden Farben sind:

0 schwarz	8 grau
1 dunkelblau	9 hellblau
2 dunkelgrün	10 hellgrün
3 zyan	11 hellzyan
4 rot	12 hellrot
5 magenta	13 hellmagenta
6 braun	14 gelb
7 weiß	15 intensivweiß

Die COLOR und SCREEN-Befehle, mit denen sich die Bildschirmeigenschaften einstellen lassen, gelten auch für einfarbige Bildschirme. Normalerweise können nur zwei Farben, schwarz und grün dargestellt werden.

GRAFIKMODUS

Der Grafikmodus ist hingegen komplizierter. Um grafische Darstellungen zu ermöglichen, setzt sich der Bildschirm für die Software aus einzelnen Punkten zusammen. (Diese Punkte bzw. Bildelemente werden oft als Bildpunkte bezeichnet). Die Anzahl der Bildpunkte pro Bildschirm legt den Grad der Auflösung fest. Mit Hilfe des SCREEN-Befehls können Sie dann zwischen drei Auflösungsmodi: niedrige, mittlere und hohe Auflösung wählen.

Niedrige Auflösung

Bei diesem Anzeigemodus besteht der Bildschirm aus 200 horizontalen Bildpunkt-Zeilen, wobei jede Zeile 320 Bildpunkte enthält. Der Bildschirm besteht an sich aus 25 Zeilen bei 40 Zeichen.

Je nachdem, ob Sie einen einfarbigen oder farbigen Bildschirm verwenden, lassen sich für den Vordergrund (Zeichendarstellung) und den Hintergrund (Bildschirm) verschiedene Farben bzw. bis zu 4 Grauwerte einstellen.

Mittlere Auflösung

Bei der mittleren Auflösung besteht der Bildschirm aus 640 Bildpunkten horizontal und 200 Bildpunkte senkrecht über dem Bildschirm. Jede der 25 Zeilen auf dem Bildschirm nimmt 80 Zeichen auf. Dieser Darstellungsmodus verwendet nur schwarz und weiß, ganz gleich ob Sie einen einfarbigen oder farbigen Bildschirm haben.

Hohe Auflösung

Wieder gibt es hier 25 Zeilen pro Bildschirm, wobei jede Zeile 80 Zeichen enthält. Bei diesem Anzeigemodus besteht der Bildschirm aus 400 horizontalen Bildpunkt-Zeilen mit je 640 Bildpunkten. Es gibt zwei verschiedene Auflösungsmodi. Bei einfarbiger Grafik mit hoher Auflösungs (SCREEN 3) erfolgt die Anzeige am Bildschirm nur einfarbig. Der farbige Grafikmodus mit hoher Auflösung (SCREEN 4) ist dem mit niedriger Auflösung ähnlich. Hier können ebenfalls nach Wunsch Farben oder auf einem einfarbigen Bildschirm bis zu vier Grauwerte ausgewählt werden.

X und Y Koordinaten

Die meistverwendete Methode, Punkte auf einer graphischen Anzeige anzusprechen ist die Verwendung von x und y Koordinaten. Dabei gilt die x-Koordinate für die horizontale Lage und die y-Koordinate für die vertikale Lage auf dem Bildschirm. 0.0 stellt die erste Bildpunktposition in der oberen linken Ecke (Ursprung) des Bildschirms dar. Gewöhnlich lassen sich die Koordinaten auf zwei Arten darstellen: in der absoluten Adressierung, wo die Koordinaten x,y die genaue Position festlegen oder die relative Form, bei der die Koordinaten x,y den relativen Wert vom zuletzt bezogenen Punkt darstellen. Bei der Festlegung der Koordinaten nach der relativen Form muß das Wort STEP einbezogen werden, um der Software mitzuteilen, daß die Adressierung relativ erfolgt. Z.B. spricht der nachfolgende GW-BASIC-Befehl einen Punkt in der Nähe des Bildschirmmittelpunkts eines Bildschirms mit hoher Auflösung an und läßt ihn aufleuchten:

```
PSET (320,199)
```

Ist der nächste Punkt, den Sie zum Aufleuchten bringen wollen, in Beziehung zum vorausgehenden Punkt bekannt (z.B., wenn Sie den Bildpunkt 6 Punkte nach rechts und 4 Bildpunkte unter dem zuletzt angesprochenen Punkt zum Aufleuchten bringen lassen wollen), können Sie folgende Adressiermethode verwenden:

```
PSET STEP (6,4)
```

Dies erspart Ihnen die Berechnung der absoluten Koordinaten in bezug auf den 0.0-Ursprung. Befindet sich der neue Punkt links oder über dem zuletzt angesprochenen Bildpunkt, sind entsprechende Minuswerte erforderlich. Besitzen Sie eine mathematische Vorbildung, werden Sie bemerken, daß bei diesem Koordinatensystem keine kartesischen Koordinaten verwendet werden. In GW-BASIC gibt es einen Befehl (WINDOW), mit dem Sie Ihr eigenes Koordinatensystem setzen können, wobei Sie ebenfalls auf Wunsch das kartesische Koordinatensystem ver-

wenden können. Die vorausgegangene Erläuterung und die nachfolgende Einführung in die Grafikmodi beziehen sich auf den Bildschirm in Form von Koordinaten, wie sie am Anfang durch GW-BASIC gesetzt wurden, nämlich mit ihrem Ursprungspunkt in der linken Ecke oben am Bildschirm und ihrem maximalen Y-Wert unten am Bildschirm.

Farbauswahl im Graphikmodus

Bei Verwendung eines farbigen Bildschirms können Sie verschiedene Farben für den Vordergrund (Zeichen oder grafische Darstellungen) und für den Hintergrund (Bildschirm) auswählen. Sie können dann zwischen zwei Paletten, die jeweils 3 Farben bezeichnet mit 1, 2 und 3 enthalten. Die Palette setzt sich aus folgenden Farben zusammen:

Palette 0	Palette 1	Farbe
Grün	Zyan	1
Rot	Magenta	2
Braun	Weiß	3

Es ist außerdem möglich, von einer Palette zur anderen überzuwechseln, worauf sich die Farben am Bildschirm in diejenigen der neu gewählten Palette verändern. Zu den Palettenfarben können Sie noch eine Farbe Ihrer Wahl der Hintergrundfarbe (Farbe 0) hinzufügen, die von der Palettenfarbumschaltung unabhängig ist und irgendeine von den 16 im Textmodus vorhandenen Farben sein kann.

Die Tatsache, daß dies ein Grafik- und kein Textmodus ist, hindert Sie nicht daran, den GW-BASIC Textmodus aufzurufen. Beim Farbbildschirm werden die Zeichen in der Farbe 3 der von Ihnen gewählten Palette geschrieben, während die Hintergrundfarbe diejenige ist, die Sie für die Farbe 0 gewählt haben.

	farbige Anzeige	einfarbige Anzeige
niedrige Auflösung SCREEN I	4 Farben	4 Grauwerte
Mittlere Auflösung SCREEN 2	schwarz/weiß	schwarz/weiß
Hohe Auflösung SCREEN 3	schwarz/weiß	schwarz/weiß
SCREEN 4	4 Farben	4 Grauwerte

Erweiterter Zeichensatz beim Grafikmodus

Wenn Sie vor eionem Aufruf von GW-BASIC den NCR-DOS Befehl GRAFTABL.COM ausführen, verfügen Sie in GW-BASIC über einen

erweiterten Zeichensatz, d.h. auch über solche Zeichen, die im herkömmlichen ACI-Code nicht vorhanden sind (z.B. ä, ö, etc.).

GRAPHIKMODUS

Bei Verwendung eines Farbgerätes können Sie mit dem SCREEN-Befehl zwischen zwei Auflösungsstufen der graphischen Bildschirmdarstellung wählen: der mittleren und der hohen Auflösung. Bei der mittleren Auflösung besteht der Bildschirm aus 200 Zeilen von je 640 Punkten. (Diese Punkte bzw. Bildelemente werden oft als „Bitpunkte" bezeichnet). Neben der Darstellung des ASCII-Standardteils des GW-BASIC-Zeichensatzes (bis zu 127 Codewerte) können Sie im Graphikmodus Zeichnungen und einzelne Punkte auf dem Bildschirm leuchtend darstellen.

Die am häufigsten verwendete Methode, Punkte in einer graphischen Darstellung anzusprechen, ist die Verwendung von X und Y-Koordinaten. Die X-Koordinate bezieht sich auf die Entfernung vom Ursprung (Ausgangspunkt) in horizontaler Richtung und die Y-Koordinate auf die Entfernung in vertikaler Richtung vom Ursprung. Die X-Koordinate wird stets als erste der beiden Koordinaten angegeben. Folglich sprechen die Koordinaten 160,100 bei Verwendung der Standardauflösung einen Punkt im mittleren Bildschirmbereich an. GW-BASIC bietet zwei verschiedene Methoden an, um einen Punkt auf dem Bildschirm anzusprechen. Sie können einen Punkt bezogen auf die linke obere Ecke des Bildschirms ansprechen. Zum Beispiel läßt folgender Befehl

```
PSET (320,2)
```

einen Punkt in der Nähe der dritten der 200 Zeilen am Bildschirm mit hoher Auflösung aufleuchten. Wenn der nächste Punkt, der aufleuchten soll, in Bezug auf den letzten Punkt bekannt ist, (z.B. wenn Sie den Punkt, der sich 6 Punkte nach rechts und 4 Punkte unter dem zuletzt angesprochenen Bildschirmpunkt befindet), können Sie folgende Adressiermethode anwenden:

```
PSET STEP (6,4)
```

Dies erspart Ihnen die Berechnung der absoluten Koordinaten bezogen auf den 0,0 Ursprung. Wenn sich der neue Punkt links oder über den zuletzt angesprochenen Bildschirmpunkt befindet, sind entsprechende Minuswerte erforderlich.

Besitzen Sie eine mathematische Vorbildunmg, werden Sie bemerken, daß bei diesem Koordinatensystem keine kartesischen Koordinaten

benützt werden. GW-BASIC beinhaltet einen Befehl (WINDOW), mit dem Sie Ihr eigenes Koordinatensystem setzen können, dabei kann auch das kartesische Koordinatensystem verwendet werden. Die vorausgegangenen Erläuterungen und die folgende Einführung in den Graphikmodus beziehen sich auf den Bildschirm in Form von Koordinaten, wie sie am Anfang von GW-BASIC gesetzt wurden, d.h. mit ihrem Ursprungspunkt in der linken Ecke oben und ihren maximalem Y-Wert unten am Bildschirm.

STANDARDAUFLÖSUNG

Es handelt sich hier um den Bildschirmmodus, bei dem der Bildschirm aus 320 x 200 Punkten besteht. Genau wie im Textmodus befindet sich der Ursprung oben in der linken Ecke des Bildschirms. Im Unterschied zum Textmodus wird der Ursprung als 0,0 (und nicht als 1,1) definiert und die erste der beiden Koordinaten bezieht sich auf die horizontale Richtung. Sie können zwischen zwei Paletten mit je drei Farben wählen, die mit 1, 2 und 3 gekennzeichnet sind. Die einzelnen Paletten enthalten folgende Farben:

Palette 0	Palette 1	Farbe
Grün	Zyan	1
Rot	Magenta	2
Braun	Weiß	3

Es ist außerdem möglich, von einer Palette auf die andere überzuwechseln, worauf sich die am Bildschirm angezeigten Farben in die der neuen Palette verwandeln. Außer den Farben der Palette können Sie der Hintergrundfarbe eine von Ihnen gewählte Farbe (Farbe 0) zuweisen, die von der Palettenumschaltung nicht betroffen ist und jede der 16 Farben des Textmodus sein kann. Die Tatsache, daß es sich hier um einen Graphik- und nicht um einen Textmodus handelt, hindert Sie nicht daran, den GW-BASIC Textmodus aufzurufen. Sie können 25 Zeilen mit je 40 Zeichen verwenden. Bei einem Farbbildschirm werden die Zeichen in der Farbe 3 der von Ihnen ausgewählten Palette geschrieben, wobei der Hintergrund die Farbe aufweist, die für die Farbe 0 gewählt wurde.

HOHE AUFLÖSUNG

Bei der hohen Auflösung stehen 200 Punkte in vertikaler und 640 Punkte in horizontaler Richtung zur Verfügung. Wie bei der Standardauflösung befindet sich der Koordinatenursprung oben links am Bildschirm und der erste Punkt wird mit 0,0 gekennzeichnet. Dementsprechend ist der Punkt unten rechts am Bildschirm mit 639,199 zu definieren.

Zwei Farben stehen bei diesem Graphikmodus zur Verfügung – schwarz und weiß. Sie haben die Kennummern 0 bzw. 1. Sie können Zeichen aus

dem GW-BASIC Textmodus am Bildschirm ausgeben. Während bei der Standardauflösung 40 Zeichen auf einer Zeile ausgegeben werden können, ist es möglich, bei hoher Auflösung bis zu 80 Zeichen darzustellen. Die einzelnen Zeichen werden in weiß auf schwarzem Hintergrund ausgegeben.

ÜBUNGEN

Die folgenden Beispiele können nur an einem Adapter mit Farbbildschirm ausprobiert werden. Das erste Beispiel zeigt die vorhandenen Farben am Bildschirm sowohl in statisch als auch in blinkend. Die einzelnen Farben mit Ausnahme von schwarz und grau werden auf schwarzem Hintergrund gezeigt. (Für eine unsichtbare Bildschirmanzeige brauchen Sie nur die gleiche Farbe für den Vorder- und Hintergrund wählen).

Dieses Programm bedient sich des Textmodus zur Anzeige am Bildschirm. Das am Bildschirm ausgegebene Zeichen hat den Codewert 219 (ein Rechteck, das den gesamten Raum für ein Zeichen ausfüllt). Die STRING$-Funktion in Zeile 10 richtet die Variable B$ mit 40 solcher Rechtecke ein. In Zeile 20 wird die Hintergrundfarbe auf schwarz und die Breite der Bildschirmanzeige auf 80 Zeichen gesetzt. Im restlichen Programm werden die Grundfarben (0 to 7) nacheinander ausgewählt. Die Grundfarbe (Zeile 50) wird zuerst angezeigt, dann wird diese blinkend (Zeile 70), intensiv (Zeile 90) und zuletzt leuchtend und blinkend ausgegeben. In jedem Fall erhält man den blinkenden Zustand durch Addition von 16 zu dem Wert, der sonst als Farbwert gilt. Die Zeilenbreite von 80 hat zur Folge, daß die 4 Versionen für jede Farbe bei der Anzeige am Bildschirm 2 Zeilen beansprucht.

```
5 SCREEN 0
10 B$=STRING$(40,CHR$(219)
20 COLOR ,0:WIDTH 80
30 FOR p%=1 TO 7
40 COLOR LP%,0
50 PRINT B$;
60 COLOR LP%+16,0
70 PRINT B$;
80 COLOR LP%+8,0
90 PRINT B$;
100 COLOR LP%+24,0
110 PRINT B$;
120 NEXT LP%
125 COLOR 7,0
130 END
```

Das folgende Programm zeigt nur einige der graphischen Möglichkeiten, die bei hoher Auflösung im Farbgrafikbildschirm zur Verfügung stehen. Das Programm führt am Bildschirm eine punktweise Zeichnung aus, wobei Sie mit der Zifferntastatur die einzelnen Punkte richtungsmäßig steuern können. Sie können bestimmen, ob die GW-BASIC-Koordinaten (bisher in diesem Kapitel verwendet) oder die richtigen kartesischen Koordinaten für die graphische Zeichnung gültig sind. Durch Änderung einer einzigen Programmzeile können Sie sogar einen anderen Punkt als Koordinatenursprung bestimmen. Mit diesem Programm können Sie die Farbpalette wechseln und eine Farbe innerhalb der gewählten Farbpalette auswählen.

Die Programmvariablen werden folgendermaßen verwendet:

Sämtliche Zahlenvariablen enthalten ganzzahlige Werte (Zeile 20).

EXX, WYE	X und Y-Koordinaten des gegenwärtig angesprochenen Punktes am Bildschirm
X1,Y1 X2,Y2	Die X und Y-Koordinaten der zwei diagonal gegenüberliegenden Bildschirmecken. Beim WINDOW-Befehl definieren diese die Anzahl der adressierbaren Punkte auf der horizontalen und vertikalen Bildschirmachse. Beim WINDOW SCREEN-Befehl bezieht sich X1,Y1 auf die obere linke Bildschirm ecke, während sich X2,Y2 auf die untere rechte Bildschirmachse bezieht. Wird das Wort SCREEN beim WINDOW-Befehl nicht eingegeben, sind die kartesischen Koordinaten gültig: X1,Y1 be deutet die untere linke Ecke und X2,Y2 die obere rechte Ecke.
PAL	Enthält 0 oder 1 für die augenblicklich angesprochene Palette
COL	Enthält 1, 2, bzw. 3 für die Farbe, die gerade aus der augenblicklich angesprochenen Palette ausgewählt wurde.
K$	Speichert ein Zeichen, das aus der Tastatur mit Hilfe der INKEY$- Funktion gelesen wurde. GW-BASIC verbleibt in der Tasta-

tur-Lese-Routine (Zeile 330 bis 360), bis eine Taste betätigt wird.

C$ — Wenn diese Variable „k" enthält, zeigt dieses Zeichen die Verwendung kartesischer Koordinaten an (Zeilen 60,160,170).

ERON$,EROFF$ — Am Anfang enthält ERON$ ein „N" für nein und EROFF$ ein „Y" für ja. Zeile 150 prüft, ob ERON$ den Wert „Y" angenommen hat. Ist dies der Fall, werden Punkte am Bildschirm gelöscht und nicht beleuchtet. Die Werte dieser beiden Variablen werden ausgetauscht (Zeile 310), sobald die Taste 5 im Verlauf der punktweisen Zeichnung betätigt wird.

B$ — Bedeutet eine Zeichenkette von 30 Leerstellen, die zum Überschreiben von am Bildschirm ausgegebenen Meldungen verwendet wird, die vom Programm in der Textzeile 25 ausgegeben wurden.

Wenn Sie kartesische Koordinaten verwenden wollen, werden Sie nach Eingabe des RUN-Befehls aufgefordert „k" einzugeben, andernfalls genügt jede beliebige Eingabe.

Die Bildschirmanzeige wird dann gelöscht und der Graphikmodus mit hoher Auflösung wird gesetzt bei aktivierter Farbdarstellung (Zeile 50). Daraufhin wird eines der Bildschirmfenster nach dem ausgewählten Koordinatensystem (Zeile 60) eingerichtet. Die für den numerischen Koordinatenbereich verwendeten Werte, die die Höhe und die Breite des Bildschirms setzen, werden von den Werten bestimmt, die in Zeile 40 den Variablen X1, Y1, X2 und Y2 zugeordnet sind. Die hier gegebenen Werte bedienen sich der maximalen Auflösung, die bei der Graphik mit hoher Auflösung zur Verfügung steht. Der Ursprung (0,0) liegt so nahe wie möglich zum Bildschirmmittelpunkt. Die alternativen Werte, die in der REMark-Zeile 30 vorgeschlagen werden, würden den Ursprung in die linke oberste bzw. unterste Ecke des Bildschirms positionieren, je nachdem ob kartesische oder SCREEN-Koordinaten verwendet werden. Sie möchten dies gerne später ausprobieren (löschen Sie einfach das Wort REM aus Zeile 30 und setzen Sie dieses Wort am Anfang der Zeile 40 ein). Sie könnten sogar eine Art horizontaler bzw. vertikaler Verzerrung durch Veränderung der Proportionen zwischen den X und Y-Werten hervorrufen.

Nachdem Sie sich versichert haben, daß die Num Lock-Taste die numerischen Funktionen (und nicht die Cursor Bewegung) aktiviert, können Sie die Tastengruppe rund um die 5 auf der numerischen Tastatur zur Beleuchtung der Punkte in 8 Richtungen (8 beleuchtet den Punkt nach Norden, 9 den nach Nordosten des letzten angesprochenen Punktes etc.) benützen. Wenn Sie von den Fensterwerten in Zeile 40 Gebrauch machen, dann befindet sich der erste angesprochene Punkt im Mittelfeld des Bildschirms. Somit können Sie sich nicht gleich am Anfang verirren. Bei Betätigung der Taste 5 bewegt sich ein Leuchtpunkt quer über den Bildschirm, je nachdem wie Sie die numerische Tastatur bedienen, dabei hinterläßt der Punkt keine Leuchtspur. Sie können zu einer neuen Zeichnungsposition übergehen. Diese Unterdrückung des Zeichnungsvorgangs bleibt solange in Kraft, bis Sie wieder Taste 5 drücken.

Die gesamte Bewegung über den Bildschirm erfolgt in den Zeilen 140 bis 300. Die PSET und PRESET-Befehle werden verwendet, um Punkte zu zeichnen bzw. zu löschen. Diese Befehle, der GOTO-Befehl und die VAL-Funktion, werden im Kapitel 4 genau beschrieben.

Die Koordinaten für den am Bildschirm gegenwärtig angesprochenen Punkt werden unten am Bildschirm im X,Y-Format (Zeile 290) ausgegeben. Nichts kann Sie daran hindern, daß Sie einmal über den Rand des Bildschirms hinausgeraten. Ein Warnton wird Ihnen mitteilen, wann dies der Fall ist (Zeile 270).

Die Hintergrundfarbe wird am Anfang auf schwarz gesetzt, die Farbe für die Zeichnung auf Magenta (Farbe 2 der Palette 1, Zeile 90). Wenn Sie, anstatt eine Zifferntaste (1 bis 9) zu betätigen, das kleine k eingeben, wird das Programm angewiesen, daß eine andere Punktfarbe (Zeile 130) gewünscht wird. Dies wird in einem Unterprogramm in Zeile 370 bis 500 ausgeführt, dabei wird eine 0 oder 1 als Palettennummer, eine Zahl 1,2 oder 3 als Farbe aus der Palette angenommen. Wenn Sie die Palette wechseln, ändert sich dementsprechend auch die Farbe der gesamten bisher erstellten Zeichnung (grün (–) zyan, rot (–), braun (–) weiß). Nach dem Farbwechsel können Sie den Zeichnungsvorgang wieder mit der numerischen Tastatur steuern.

Wenn Sie beim Steuern des Zeichnungsvorgangs das kleine x eingeben, dann springt GW-BASIC nach Zeile 510, worauf am Bildschirm der Textmodus herausnehmen! wiederhergestellt wird. Sie haben auch die Möglichkeit, eine Auswahl von weiteren graphischen Funktionen an diesem Punkt einzufügen, z.B. CIRCLE-Zeichnung (Kreiszeichnung) oder Farbbelegung einer Bildschirmfläche. Sie könnten Ihre Zeichnung sogar in einer Feldvariablen und schließlich auf Platte speichern. In Kapitel 4 finden Sie sämtliche Einzelheiten, die für eine optimale Nutzung der graphischen Möglichkleiten von GW-BASIC erforderlich sind.

Kartesische oder SCREEN Koordinaten?

```
10 INPUT "k für kartesische";C$
```

Alle Zahlenvariablen ganzzahlig

```
20 DEFINT A-Z
```

WINDOWs für den Ursprung in der Ecke und im Zentrum

```
30 REM X1=0:y1=0:X2=639:Y2=399
40 X1=320:Y1=200:X2=319:Y2=199
```

Hohe Auflösung, Farbe aktiviert

```
50 SCREEN 4,0
```

WINDOW setzen, je nachdem ob es sich um kartesische oder nichtkartesische Koordinaten handelt.

```
60 IF C$<>"k" THEN WINDOW SCREEN(X1,Y1)-(X2,Y2):
   ELSE WINDOW (X1,Y1)-(X2,Y2)
```

Am Anfang werden die Leuchtpunkte nicht gelöscht

```
70 ERON$="N":EROFF$="Y"
```

Zeichenkette von 30 Leerzeichen, die zum Löschen von „Palette?" und „Farbnummerß?" benützt werden

```
80 B$=STRING$(30," ")
```

Schwarzer Hintergrund, Palette 1, die anfangs aktive Zeichnungsfarbe ist Magenta

```
90 COLOR 0,1:COL=2
```

Der erste angesprochene Punkt ist der Ursprung

```
100 EXX0=0:WYE=0
```

Nach jeder Bewegung am Bildschirm bzw. Farbeinstellung kehrt GW-BASIC hierher zurück.

```
110 GOSUB 330
120 IF K$="x" THEN GOTO 510
130 IF K$="c" THEN GOSUB 370:GOTO 110
140 IF K$("1" OR K$)"9" THEN GOTO 110
150 IF ERON$="Y" THEN PRESET (EXX,WYE)
```

```
160 IF C$<>"c" THEN ON VAL(K$) GOTO 180,190,200,210,
    220,230,240,250,260
170 IF C$="c" THEN ON VAL(K$) GOTO 240,250,260,210,
    220,230,180,190,200
180 EXX=EXX-1:WYE=WYE+1:GOTO 270
190 WYE=WYE+1:GOTO 270
200 EXX=EXX+1:WYE=WYE+1:GOTO 270
210 EXX=EXX-1:GOTO 270
220 GOTO 310
230 EXX=EXX+1:GOTO 270
240 EXX=EXX-1:WYE=WYE-1:GOTO 270
250 WYE=WYE-1:GOTO 270
260 EXX=EXX+1:WYE=WYE-1:GOTO 270
270 IF EXX>X2 OR EXX<X1 OR WYE<Y2 OR WYE>Y1
    THEN BEEP
280 PSET (EXX,WYE),COL
```

Geben Sie die augenblicklich angesprochenen Koordinaten aus und kehren Sie zur nächsten Eingabe auf der Tastatur zurück.

```
290 LOCATE 25,1:PRINT EXX;", ";WYE;
300 GOTO 110
```

GW-BASIC gelangt erst hierher, wenn Taste 5 betätigt wurde

```
310 SWAP ERON$,EROFF$
320 GOTO 110
```

Unterprogramme

```
330 REM***** Tastatur lesen
340 K$=INKEY$
350 IF K$="" THEN GOTO 340
360 RETURN
370 REM ***** SET COLOR
380 LOCATE 25,1
390 PRINT "Palette?";
400 GOSUB 330
410 IF K$<"0" OR K$>"1"THEN GOTO 380
420 PAl%=Val(K$)
430 LOCATE 25,1
440 PRINT "Farbnummer?";
450 GOSUB 330
460 IF K$<"1" OR K$>"3" THEN GOTO 430
470 COL%=VAL(K$)
```

```
480 COLOR, PAL
490 LOCATE 25,1:PRINT B$;
500 RETURN
```

Wenn Sie während des Zeichenvorgangs x eingeben, gibt Gw-BASAIC folgendes aus:

```
510 REM ***** Farbfunktionen?
520 SCREEN 0
530 END
```

GW-BASIC-Befehle und Funktionen

Dieses Kapitel enthält eine ausführliche Beschreibung aller Befehle und Funktionen.

Befehle sind Anweisungen an GW-BASIC, die entweder im direkten Modus (unmittelbare Tastatureingabe), oder im indirekten Modus (innerhalb eines Programms) ausgeführt werden.

Eine Funktion tauscht mittels einer festen Formel einen Wert in irgendeinen anderen Wert um. Die in diesem Kapitel beschriebenen Funktionen sind „eingebaut", das heißt, in GW-BASIC schon vorhanden. Diese Funktionen können von jedem Programm ohne weitere Definition abgerufen werden. GW-BASIC kann das Ergebnis einer Funktion nicht selbst auswerten. Es benötigt dazu noch einen Befehl, der ihm mitteilt, was mit dem Ergebnis geschehen soll. Das Ergebnis kann zum Beispiel am Bildschirm angezeigt, an den Drucker weitergegeben, und einer Variablen zugewiesen werden. Die Variable einer Funktion, das heißt, der Wert den die Funktion verwenden soll, steht bei GW-BASIC immer in „<>"-Klammern.

Wenn Sie einen Fließkommawert für eine Funktion angeben, bei der ein Ganzzahlwert erforderlich ist, rundet GW-BASIC die Stellen hinter dem Komma, und verwendet den resultierenden Ganzzahlwert. Geben Sie beim Laden von GW-BASIC die /D-Option an, so werden die Funktionen ATN, COS, EXP, LOG, SIN, SQR und TAN auf „doppelte Genauigkeit" (double precision) berechnet. Sonst wird „einfache Genauigkeit" (single precision) verwendet. Für weitere Informationen über mathematische Funktionen, die nicht in GW-BASIC enthalten sind, siehe Anhang D.

Die folgenden Seiten bieten eine kurze Übersicht über sämtliche GW-BASIC-Befehle und Funktionen. Diese sind zunächst in Gruppen aufgelistet. Jeder Gruppe ist eine Überschrift zugeordnet, welche die Verwendung der darunter aufgelisteten Befehle und Funktionen bezeichnet.

Jeder Befehl sowie jede Funktion ist kurz beschrieben, ohne auf Syntaxdetails einzugehen. Wenn Sie also GW-BASIC-Anfänger sind, werden Sie auf einen Blick in der Lage sein, die GW-BASIC-Funktionen zu finden, die einer ganz bestimmten Programmiersituation entsprechen. Sie

können dann zu den vollständigen Beschreibungen übergehen, die im Hauptteil dieses Kapitels enthalten sind.

Die Überschriften sind, der Reihenfolge nach:

GW-BASIC-Verwaltung
Das Editieren von Programmen
Das Laden und Speichern von Programmen
Dateiverarbeitung
Tastatur und andere „Nicht-Platten"-Eingaben
Zeichen auf Bildschirm und Drucker
Graphik
Der Lautsprecher
Programmvariablen und Zeichenumwandlung
Behandlung von Zeichenketten
Mathematische Funktionen
Bedingte Sprünge
Programmunterbrechung
Weitere Befehle und Funktionen

Einige vielseitige Befehle und Funktionen erscheinen unter mehr als einer Überschrift.

GW-BASIC-Verwaltung

CLEAR	Löscht Programmvariablen und begrenzt wahl weise den für GW-BASIC zur Verfügung stehen den Speicherraum
CONT	Fährt mit dem Ablauf eines Programms nach einer Unterbrechung (break) fort
DEF SEG	Definiert ein Speichersegment im Arbeitsspeicher
DEF USR	Definiert die Startadresse eines Maschinenprogramms
END	Das Programm hält an, alle Dateien sind geschlossen, „Ok" wird angezeigt
FRE	Eine Funktion die den noch freien, momentan von GW-BASIC nicht benötigten Speicherplatz, angibt

NEW	Löscht Programme und Ihre Variablen aus dem Speicher, beeinflußt GW-BASIC aber nicht
RANDOMIZE	Setzt den Ausgangspunkt für den Zufallszahlengenerator, der von der RND-Funktion verwendet wird
RUN	Lädt und beginnt mit einem Programmablauf, oder beginnt mit dem Ablauf eines bereits geladenen Programmes ab einer vorher bestimmten Zeile
SHELL	Ausführung einer NCR-DOS-Befehlsdatei und Rückkehr zu GW-BASIC
STOP	Beendigung des Programmablaufs und Bildschirmanzeige der Zeilennummer bei der das Programm beendet wurde
SYSTEM	Schließt alle Dateien und kehrt zur NCR-DOS-Systemebene zurück
TRON, TROFF	Programmablaufanalyse ein- und ausschalten.
VARPTR$	Eine Funktion, die eine spezifizierte Variablenadresse an den Arbeitsspeicher gibt. Diese Information wird manchmal von Maschinensprachprogrammen benötigt.
WAIT	GW-BASIC hält das Programm an, bis ein bestimmter Wert einen spezifizierten Port erreicht.

Das Editieren von Programmen

AUTO	Erzeugt Programmzeilennummern automatisch und erpart Ihnen damit deren Eingabe
DELETE	Löscht eine Programmzeile oder mehrere aufeinanderfolgende Programmzeilen

EDIT	Gibt eine Programmzeile zum Editieren am Bildschirm aus
LIST	Listet Programmzeilen am Bildschirm auf, oder leitet sie an eine Datei weiter
LLIST	Wie LIST, jedoch werden die Programmzeilen auf dem Drucker ausgegeben
NEW	Löscht den Arbeitsspeicher, einschließlich aller Programme, außer GW-BASIC
RENUM	Numeriert Programmzeilen neu
KEY	Belegt eine Funktionstaste auf der Tastatur
KEY ON/OFF/LIST	Schaltet die Anzeige der Funktionstastenbelegung ein und aus.

Das Laden und Speichern von Programmen

BLOAD	Lädt Binärdaten in den Speicher. Wird besonders für Maschinenprogramme verwendet.
BSAVE	Speichert Binärdaten auf Platte
LOAD	Lädt eine Programmdatei von der Platte. Wahlweise wird das Programm sofort ausgeführt
MERGE	Lädt ein Programm von der Platte und verbindet es mit einem Programm, das sich schon im Speicher befindet
SAVE	Speichert ein Programm auf Platte, wahlweise in gesichertem Format

Dateiverarbeitung

CHDIR	Ändert das aktuelle Inhaltsverzeichnis
CLOSE #	Schließt eine Datei für den Programmzugriff

ENVIRON	Änderung der Systemparameter des Betriebssystems
EOF	Eine Funktion, die darüber informiert, ob das Dateiende erreicht ist.
GET #	Liest einen Datensatz von einer Datei mit wahlfreiem Zugriff
FIELD #	Definiert ein Feld in einem Puffer für wahlfreien Zugriff auf Dateien, und kann so verschiedene Teile eines Datensatzes verschiedenen Programmvariablen zuweisen
FILES	Sucht im Platteninhaltsverzeichnis nach einer bestimmten Datei oder Gruppe von Dateien, und wenn auffindbar, zeigt den (die) Dateinamen an
INPUT #	Liest Daten von einer Datei und weist diese Daten sofort einer oder mehr Variablen zu
KILL	Löscht eine Plattendatei
LINE INPUT #	Liest eine ganze Zeile von einer Datei
LOC	Erteilt Informationen über den aktuellen Stand der Bearbeitung einer Datei mit wahlfreiem Zugriff, einer sequentiellen Datei oder einer Textübertragungsdatei
LOF	Informiert über die Länge einer angegebenen Datei
MKDIR	Erstellt ein Inhaltsverzeichnis
NAME...AS...	Benennt eine Plattendatei um
OPEN	Ein vielseitiger Befehl, der zum Öffnen einer OPEN..FOR..AS Datei für den Programmzugriff verwendet wird OPEN „COMEine Sonderform des OPEN-Befehls, der eine Datei für die Datenübertragung öffnet

PRINT #	Schreibt eine Liste von Ausdrücken oder Daten PRINT #..USING auf eine Datei
PUT	Schreibt Daten von einem Pufferspeicher für wahlfreien Dateizugriff in eine Datei
RESET	Schließt alle Plattendateien
RMDIR	Löscht das angegebene Inhaltsverzeichnis
VARPTR(#)	Teilt die Adresse des Dateisteuerblocks einer Datei mit
WRITE #	Lenkt die Datenausgabe auf eine bestimmte Datei

Tastatur und andere „Nicht-Platten"-Eingaben

DATA	Eine Auflistung von Datengruppen, die durch den READ-Befehl sequentiell gelesen werden
INKEY$	Liest ein Zeichen von der Tastatur
INP	Diese Funktion gibt den Wert eines Bytes aus, das von einem bestimmten Port des Computers gelesen wurde. Die Ausgabe von Daten an einen Port erfolgt mit Hilfe des OUT-Befehls
INPUT	Liest die Eingabe von der Tastatur in eine Variable
KEY	Belegt eine Funktionstaste
KEY ON/OFF/LIST	Schaltet die Anzeige der Funktionstastenbelegung ein und aus
LINE INPUT	Übernimmt eine Eingabezeile von der Tastatur, wobei Kommas und ähnliche Abgrenzungszeichen nicht beachtet werden

ON KEY...GOSUB	GW-BASIC überträgt die Programmsteuerung unter Angabe der Anfangszeile an eine Unterroutine, wenn eine bestimmte Cursor-Steuertaste oder Funktionstaste betätigt wird
PEN	Gibt die Lichtgriffel-Koordinaten an
PEN ON/OFF/STOP	Schaltet die Lichtgriffel-Funktion ein und aus
READ	Liest den nächsten Wert von der DATA-(Daten)-Liste in eine Variable
RESTORE	Bewirkt, daß DATA-Zuweisungen in einem Programm erneut durch READ ab einer bestimmten Zeile gelesen werden
STICK	Gibt die Koordinaten an, die von einem Joystick (Steuerknüppel) übertragen werden
STRIG	Diese Funktion überprüft, ob eine Joystick-Taste betätigt wird oder betätigt wurde
STRIG ON/OFF	Sperren oder Freigeben einer Joystick-Taste

Zeichen auf Bildschirm oder Drucker

CLS	Löscht den Bildschirm
COLOR	Stellt die Farbe für die numerische Eingabe, die Hintergrundfarbe und die Farbe der Eingrenzung bei Farbbildschirmen ein
CSRLIN	Gibt die Bildschirmzeilennummer (1...25) an, in der sich der Cursor befindet
LCOPY	Gibt den Bildschirminhalt am Drucker aus
LOCATE	Schaltet den Cursor ein oder aus und bestimmt seine Position
LPOS	Gibt die momentane Position des Druckkopfes an
LPRINT	Druckt Daten aus
LPRINT USING	Wie LPRINT, gibt aber zusätzlich ein Druckformat an
POS	Teilt mit in welcher Bildschirmspalte sich der Cursor befindet
PRINT	Zeigt Daten auf dem Bildschirm an
PRINT USING	Wie PRINT, gibt aber zusätzlich ein Ausgabeformat an
SCREEN	Setzt unter anderem den Textmodus der Bildschirmausgabe. Als Funktion teilt SCREEN das Zeichen, oder die Farbe an der angegebenen Bildschirmposition mit
SPC	Druckt eine bestimmte Anzahl von Leerräumen in einem PRINT-Befehl
TAB	Bewegt die Druckposition oder die Bildschirm-(Cursor)-Position zur angegebenen Spalte.
WIDTH	Setzt die Breite für die Ausgabe am Bildschirm oder Drucker

WRITE	Ist ähnlich dem Print-Befehl.

Siehe auch „Das Editieren von Programmen"

Graphik

CIRCLE	Zeichnet einen Kreis mit angegebenem Mittelpunkt und Radius, einen Bogen oder auch eine Ellipse
COLOR	Wählt eine der beiden Farbpaletten und die Hintergrundfarbe
DRAW	Zeichnet eine vom Benutzer angegebene Figur auf den Bildschirm
GET	Liest die Farben aller Punkte innerhalb eines angegebenen Rechtecks in eine Feldvariable
LINE	Zeichnet Linien auf den Bildschirm und füllt, wenn erwünscht, eine abgegrenzte Fläche aus
PAINT	Füllt einen abgegrenzte Fläche auf dem Bildschirm mit einer angegebenen Farbe aus
PMAP	Setzt die von Ihrem Programm definierten „Welt"-Koordinaten in darstellbare Bildschirmkoordinaten um
POINT	Nennt die Farbe eines angegebenen Punktes
PRESET	Setzt einen angegebenen Bildschirmpunkt in die Hintergrundfarbe zurück
PSET	Läßt einen Punkt am Bildschirm in einer bestimmten Farbe aufleuchten
PUT	Liest Graphikinformationen von einer Feldvariablen und überträgt sie auf den Bildschirm

SCREEN	Setzt unter anderem den Graphikmodus
VIEW	Beschränkt oder übermittelt Änderungen auf einen bestimmten Bereich am Bildschirm
WINDOW	Definiert Bildschirmkoordinaten neu

Der Lautsprecher

BEEP	Der Lautsprecher erzeugt einen Piep-Ton
ON PLAY	Ständige Hintergrundmusik während der Ausführung eines Programmes
PLAY	Für Musiker. Als Funktion gibt es die Anzahl der noch verbleibenden Noten im Musik-Puffer an
SOUND	Für weniger Musikbegabte

Programmvariablen und Zeichenumwandlung

ASC	Gibt den ASCII-Code für das erste Zeichen einer Zeichenkette an
CDBL,CSNG	Setzt einen numerischen Wert in „einfache Genauigkeit" oder „doppelte Genauigkeit" um
CINT	Wechselt einen numerischen Wert durch Runden in eine Ganzzahl um (nicht durch Abschneiden)
CLEAR	Löscht Programmvariablen
CHAIN	Die Steuerung wird an ein anderes Programm übergeben, wobei wahlweise alle Variablen dem neuen Programm zur Verfügung gestellt werden können
COMMON	Stellt ausgewählte Variablen dem durch CHAIN verketteten Programm zur Verfügung
CVI,CVS,CVD	Setzt eine Darstellung in Zeichenform in eine Ganzzahl oder in eine Zahl „einfacher" oder „doppelter Genauigkeit" um
DEFNIT,DEFSG, DEFDBL,DEFSTR	Definiert eine oder mehrere Variablen als Ganzzahl, als Zahl mit „einfacher Genauigkeit", „doppelter Genauigkeit", oder Zeichenkette
ERASE	Löscht Feldvariablen in einem Programm
FIX	Diese Funktion wandelt eine Dezimalzahl durch Abrunden in eine Ganzzahl um
DIM	Dient zur Angabe der maximalen Anzahl von Elementen von Gruppenvariablen und der Bereitstellung des entsprechenden Speicherplatzes

INT	Wie FIX, mit der Ausnahme, daß negative Zahlenauf den nächstniedrigen Ganzzahlwert gesetzt werden (z.B. –3.67 wird –4) HEX$Tauscht einen numerischen Wert in eine hexadezimale Zeichenkette um
LET	Teilt einer Variablen einen Wert zu
MKI$,MKS$,MKD$	Setzt eine Ganzzahl oder eine Zahl mit „einfacher" oder „doppelter Genauigkeit" in eine Zeichenkette um
OCT$	Setzt einen numerischen Wert in eine oktale Zeichenkette um
OPTION BASE	Gibt an, ob 0 oder 1 der niedrigste Indexwert für eine Feldvariable sein soll
SWAP	Vertauscht die Werte zweier Variablen miteinander
STR$	Setzt einen Zahlenwert in die entsprechende ASCII-Zeichenkette um
VAL	Setzt eine ASCII-Zeichenkette in den entsprechenden Zahlenwert um
VARPTR	Gibt die Speicheradresse einer Variablen an

Behandlung von Zeichenketten

ASC	Gibt den ASCII-Code für das erste Zeichen in der Zeichenkette an
CHR$	Gibt den entsprechenden Zahlenwert für ein ASCII-Zeichen an
DEF FN	Zur Definierung Ihrer eigenen Verarbeitungsroutinen für Zeichenketten
INSTR$	Sucht eine Zeichenkette nach einer bestimmten Buchstabenfolge ab

LEFT$	Liefert den ersten Teil einer Zeichenkette
LEN	Gibt die Länge einer Zeichenkette an
LSET	Rückt eine Zeichenkette nach links auf
MID$	Ein Befehl und eine Funktion, die verwendet wird, um einen Teil einer Zeichenkette herauszuziehen oder zu ersetzen
RIGHT$	Liefert den letzten Teil einer Zeichenkette
RSET	Rückt eine Zeichenkette nach rechts auf
SPACE$	Gibt eine Anzahl von Leerräumen in einer Zeichenkettenvariablen an
STRING$	Liefert eine Zeichenkette, bestehend aus einem Zeichen oder dem ersten Zeichen einer Zeichenkettenvariablen, das je nach Angabe beliebig oft wiederholt wird
STR$	Setzt einen Zahlenwert in die entsprechende ASCII-Zeichenkette um
VAL	Setzt eine ASCII-Zeichenkette in den entsprechenden Zahlenwert um

Mathematische Funktionen

ABS	Gibt den absoluten Wert einer Zahl an
ATN	Gibt den Arcustangens im Bogenmaß an
COS	Gibt den Cosinus eines Winkels im Bogenmaß an
DEF FN	Ermöglicht die Definierung Ihrer eigenen mathematischen Funktionen
EXP	Exponentialfunktion einstellbaren Grades
FIX	Diese Funktion wandelt eine Dezimalzahl durch Abrunden in eine Ganzzahl um
INT	Wie FIX, mit der Ausnahme, daß negative Zahlen auf den nächstniedrigeren Ganzzahlwert abgerundet wird (z.B. –3.67 wird – 4)
LOG	Der natürliche Logarithmus einer Zahl
RND	Gibt eine Zufallszahl an (siehe auch RANDOMIZE)
SGN	Das Vorzeichen einer Zahl
SIN	Der Sinus eines Winkels im Bogenmaß
SQR	Die Quadratwurzel einer Zahl
TAN	Der Tangens eines Winkels im Bogenmaß

Bedingte Sprünge

CALL	Überträgt die Programmsteuerung auf ein Maschinenprogramm
FOR..TO..STEP	Wiederholt die Programmzeilen bis zum NEXT-Befehl so oft wie angegeben

GOSUB	Überträgt die Programmsteuerung an die Unterroutine, die bei einer bestimmten Programmzeile beginnt. Die Unterroutine sollte mit dem RETURN-Befehl abgeschlossen werden. Siehe auch ON GOSUB in der ausführlichen Beschreibung
GOTO	Überträgt die Programmsteuerung an eine angegebene Programmzeile. Siehe auch ON GOTO in der ausführlichen Beschreibung
IF..THEN..ELSE	Wenn die angegebene Bedingung erfüllt ist, führt GW-BASIC einen spezifizierten Befehl oder Befehle aus. Wahlweise können Sie zusätzlich festsetzen, was GW-BASIC tun soll, wenn die Bedingung nicht erfüllt ist.
NEXT	Setzt den Programmablauf am Beginn der „FOR..TO..STEP"-Schleife fort, bis die vorgeschriebene Anzahl von Durchläufen ausgeführt wurde
RETURN	Beendet eine Unterroutine und gibt die Programmsteuerung an die Zeile nach dem letzten GOSUB-Befehl, oder eine andere angegebene Zeile, zurück
USR	Zeigt den Wert einer Maschinenroutine an (ähnlich dem CALL-Befehl)
WHILE..WEND	Schließt eine Sequenz von Programmzeilen zum wiederholten Ablauf ein, solange eine angegebene Bedingung erfüllt ist

Programmunterbrechung

COM ON/OFF/STOP	Sperren und Freigeben des Datenübertragungsvorganges
ERL	Teilt die Zeilennumer mit, bei der GW-BASIC den letzten Fehler gefunden hat

ERR	Gibt die GW-BASIC-Codenummer des zuletzt gefundenen Fehlers an (Die GW-BASIC-Fehlercodes sind in Anhang A aufgeführt)
ERROR	Täuscht einen Fehler vor. Nützlich für das Testen von Fehlerroutinen.
KEY ON/OFF/STOP	Sperren und Freigeben der Programmunterbrechung für Funktionstasten und Pfeiltasten
ON COM GOSUB	Spezifiziert die Programmzeilennummer einer Unterroutine, an welche die Programmsteuerung beim Datenübertragungsvorgang übergeben wird
ON ERROR GOTO	Findet GW-BASIC einen Fehler, wird die Programmsteuerung an die angegebene Zeile übertragen
ON KEY GOSUB	Wird eine spezifizierte Funktionstaste oder Pfeiltaste betätigt, so erfolgt eine Übertragung der Programmsteuerung an die Unterroutine in der angegebenen Programmzeile
ON PEN GOSUB	Bei Benutzung des Lichtgriffels (light pen), wird die Programmsteuerung in der angegebenen Programmzeile an die Unterroutine übergeben
ON STRIG GOSUB	Bei Betätigung einer Joystick-(Steuerknüppel)-Taste, wird die Programmsteuerung in der angegebenen Programmzeile an die Unterroutine übergeben
ON TIMER	Setzt die Zeile fest, an welche die Programmsteuerung nach Ablauf einer angegebenen Zeitspanne übergeben wird (siehe auch TIME$ und TIMER in dem Abschnitt „Weitere Befehle und Funktionen“)
PEN ON/OFF/STOP	Lichtgriffel ein- und ausschalten

RESUME	Nach einer Fehlerbehandlung wird die Programmsteuerung an die Zeile zurückgegeben, in welcher der Fehler auftrat, oder in die folgende oder eine andere angegebene Zeile
STRIG ON/OFF/STOP	Sperren und Freigeben der Joystick-Taste oder der Programmunterbrechung

Weitere Befehle und Funktionen

DATE$	Gibt das Datum in den Computer ein
OUT	Überträgt ein Byte in einen Port des Computers
PEEK	Gibt den Inhalt eines Bytes an einer spezifizierten Adresse im Arbeitsspeicher an
POKE	Schreibt ein Byte an einer angegebenen Adresse in den Arbeitsspeicher
REM	Kennzeichnet einen Kommentar vom Programmierer
TIME$	Ein Kommando und eine Funktion zum Einstellen und Ablesen der Zeit
TIMER	Gibt die Zeit in Sekunden entweder seit dem letzten Systemneustart oder dem letzten Nulldurchgang des Zeitzählers (midnight) an. (Ein Systemneustart wird durch Einschalten des Computers oder die Tastenkombination Control-Alt-Del erreicht.)

Systemkompatibilität

Eine spezielle Eigenschaft Ihres NCR PC ist die Kompatibilität mit dem IBM PC/PCXT. Diese schließt auch das Programm GW-BASIC mit ein, das Programme, die mit dem „Advanced Basic" eines IBM Computers erzeugt wurden, direkt übernehmen kann.

Bei der bereitgestellten GW-BASIC-Version handelt es sich um die Version 2.0. Sie ist vollkommen kompatibel mit dem IBM Advanced Basic 2.0. Dies bedeutet, daß Sie mit GW-BASIC Programme sowohl auf dem NCR PC als auch auf dem IBM PC verwenden können, die unter IBM Advanced Basic 2.0 oder einer früheren Version geschrieben wurden. Genauso können auch mit GW-BASIC geschriebene Programme für IBM Advanced Basic 2.0 verwendet werden.

Wenn Sie GW-BASIC zum Schreiben von Programmen verwenden, die auch auf einer früheren Version von IBM Advanced Basic ausführbar sein sollen, so sollten Sie die Dokumentation dieser Version beachten. Eine Anzahl von GW-BASIC-Funktionen werden von diesen früheren Versionen nicht unterstützt. Besondere Erweiterungen von GW-BASIC 2.0 sind:

- Die Neuzuweisung des Standardeingabegerätes (INPUT, LINE INPUT) und des Standardausgabegerätes (PRINT) kann in der NCR-DOS-Kommandozeile erfolgen, die GW-BASIC lädt.

- Die selbe NCR-DOS-Kommandozeile erlaubt mit Hilfe der /M-Option die Angabe der maximalen Blockgröße, um Speicherplatz für Maschinensprachroutinen zu reservieren.

- Ausführung von NCR-DOS-Befehlsdateien innerhalb von GW-BASIC (SHELL).

- Kommunikation mit vom Benutzer angeschlossenen Geräten.

- Graphikerweiterungen:

  Abschneiden von Linien bei den Befehlen CIRCLE, LINE, PAINT, POINT, PRESET und PSET beim Verlassen des Bildschirmbereichs, um ein Umklappen auf einen anderen Bildschirmteil zu vermeiden.

  Neue Graphikfunktionen: PMAP, WINDOW und VIEW.

  Erweiterungen bei: DRAW (Winkeldrehung, Ausmalen von Graphik), LINE (die Style-Option erlaubt die Verwendung von gestrichelten und gepunkteten Linien usw.), PAINT (die Hintergrund-

option für leichteres Bemalen von Teilflächen) und POINT (unterscheidet zwischen physikalischen und „Welt"-Koordinaten).

- Andere neue Funktionen: PLAY und TIMER

- Eine neue Option (/D) in der Kommandozeile erlaubt das Berechnen von ATN, COS, EXP, LOG, SIN, SQR und TAN mit doppelter Genauigkeit. Der Zufallszahlengenerator (RANDOMIZE) kann ebenfalls mit doppelter Genauigkeit arbeiten.

- Die parity-Prüfung (Prüfung auf Übertragungsfehler) kann freigegeben oder gesperrt werden (PE-Option in OPEN „COM).

- Klangfunktion: PLAY (zusätzliche Angabe der Oktave möglich)

- Neue Funktionen für Unterbrechungroutinen: ON PLAY, ON TIMER. ON KEY ermöglicht nun den Unterbrechngsrotinenaufruf von bis zu 6 vom Benutzer definierbaren Tasten.

- Bearbeitung von Dateien:

  GET und PUT gestatten Datensatznummern bis 16.777.215, die eine nützliche Einrichtung für große Dateien mit kleinen Datensätzen darstellen.

  LOF gibt die tatsächliche Anzahl der in der Datei befindlichen Bytes an.

  EOF kann zur Neuzuordnung des Standardeingabegerätes angewandt werden.

  Neue Befehle (ENVIRON, MKDIR, CHDIR und RMDIR) ermöglichen die Manipulation von NCR-DOS-Pfaden (paths) und den Zugriff auf andere Inhaltsverzeichnisse.

- Der DELETE-Befehl: wenn keine Zeilennummer nach dem Bindestrich angegeben ist, wird das restliche Programm bis zum Ende gelöscht.

- Die Zeichen <, >, und \ sind nicht als Teil eines Dateinamens oder einer Erweiterung erlaubt, da sie nun bei der Neuzuweisung von Ein- oder Ausgabegeräten (<, >), oder zum Spezifizieren eines Pfades (\) verwendet werden können.

Die Syntaxbezeichnung

Die Beschreibungen der GW-BASIC-Befehle und Funktionen verwenden die folgende Bezeichnung zur Erklärung der „Regeln", welche die Schreibweise des Befehls oder der Funktion festlegen:

[]	Eckige Klammern zeigen an, daß der enthaltene Eintrag wahlweise möglich ist.
<>	Spitze Klammern weisen auf vom Benutzer einzugebende Daten hin. Wenn die spitzen Klammern kleingeschriebenen Text enthalten, soll der Anwender einen durch den Text definierten Eintrag vornehmen. Enthalten die spitzen Klammern Großbuchstaben, dann soll der Anwender eine so beschriebene Taste drücken. Zum Beispiel: <RETURN>.
{}	Geschweifte Klammern machen darauf aufmerksam, daß der Benutzer die Wahl zwischen einem oder mehreren Einträgen hat. Es sollte jedoch mindestens einer der Einträge in geschweiften Klammern gewählt werden, außer wenn die Einträge zusätzlich in eckigen Klammern stehen.
|	Senkrechte Striche trennen verschiedene Wahlmöglichkeiten innerhalb von geschweiften Klammern voneinander. Mindestens ein durch Striche getrennter Eintrag sollte gewält werden, es sei denn daß diese auch in eckigen Klammern stehen.
...	Drei aufeinanderfolgende Punkte zeigen an, daß ein Eintrag so oft wie benötigt oder gewünscht wiederholt werden kann.
	Großbuchstaben weisen auf Teile von Funktionen und Befehlen hin, die genau wie gezeigt eingegeben werden sollen.

Alle anderen Satzzeichen, wie Komma, Doppelpunkt, Schrägstrichzeichen und Gleichheitszeichen sollen ebenfalls genau wie gezeigt eingegeben werden.

Jede Beschreibung in diesem Kapitel ist wie folgt aufgebaut:

Syntax: Zeigt den richtigen Syntax für den Befehl oder die Funktion. Siehe Kapitel 1 in diesem Handbuch hinsichtlich Syntax-Beschreibungen.

Wird der Begriff „dateispez" als eine Option im Syntax benutzt, so bezieht er sich auf eine Kombination von Gerätebezeichnung und Dateiname, im richtigen Format für das Betriebssystem.

Verwendung: Gibt an, wofür der Befehl oder die Funktion verwendet wird.

Bemerkung: Beschreibt in Einzelheiten wie die Funktion oder der Befehl benutzt wird.

Beispiel: Zeigt Beispielprogramme oder Progammabschnitte, die die Verwendung der jeweiligen Funktion oder des Befehls deutlich machen.

Hinweis: Beschreibt Sonderfälle oder stellt zusätzliche wichtige Informationen zur Verfügung.

ABS Funktion

Syntax: ABS(X)

Verwendung: Ergibt den absoluten Wert des Ausdruckes X.

Beispiel:
```
PRINT ABS(7*(-5))
errechnet
 35
```

ASC Funktion

Syntax: ASC(X$)

Verwendung: Ergibt einen numerischen Wert, der dem ASCII-Code des ersten Zeichens des Textes X$ entspricht. (Siehe Anhang B, ASCII-Codes).

Bemerkung: Wenn X$ leer ist, wird die Fehlermeldung "Illegal function call" (unzulässiger Funktionsaufruf) ausgegeben

Beispiel:

```
10 X$=„TEST“
20 PRINT ASC(X$)
errechnet
 84
```

Dieser Wert entspricht dem ASCII-Code für den Großbuchstaben T. Für Einzelheiten über die Umwandlung von ASCII-Werten in Zeichen siehe CHR$-Funktion und das Beispiel in der Beschreibung von CONT.

ATN Funktion

Syntax: ATN(X)

Verwendung: Ergibt den Arcustangens von X im Bogenmaß. Das Ergebnis liegt im Bereich –pi/2 bis pi/2.

Bemerkung: Der Ausdruck X kann von beliebigem numerischen Typ sein. Das Ergebnis wird mit „einfacher Genauigkeit" ermittelt, wenn Sie beim Laden von GW-BASIC die /D-Option nicht angeben.

Beispiel:

```
10 INPUT X
20 PRINT ATN(X)
```

Wenn Sie 3 eingeben, wird der ausgegebene Wert 1.249046 sein.

Hinweis: Zur Umwandlung von Grad in Maß:

DEGREES = RADIANS*180/PI

wobei PI (single precision/einfache Genauigkeit) den Wert 3.141593 hat.

AUTO Befehl

Syntax: AUTO [<zeilennummer>[,<abstand>]]

Verwendung: Dient zur automatischen Generierung von Zeilennummern, während der Eingabe von Programmtext.

Bemerkung: AUTO beginnt mit der Numerierung bei <zeilennummer> und erhöht jede nachfolgende Zeile um <abstand>. Werden keine Werte angegeben, so nimmt GW-BASIC als Standardwert 10 an. Wird auf <zeilennummer> zwar ein Komma, jedoch kein <abstand> eingegeben, so wird der Wert <abstand> vom zuletzt eingegebenen AUTO-Befehl übernommen.

Falls AUTO eine Zeilennummer generiert, die bereits vorhanden ist, wird nach der Zeilennummer ein Stern angezeigt, um den Programmierer vor dem versehentlichen Überschreiben einer bestehenden Programmzeile zu warnen. Wird jedoch unmittelbar nach dem Stern die Abschlußtaste betätigt, so bleibt die ursprüngliche Programmzeile erhalten, und die nächste Zeilennummer wird generiert.

Wird der Cursor in eine andere Zeile auf dem Bildschirm bewegt, fährt die Numerierung dort fort.

AUTO wird mit der Control-Break-Taste beendet. Die Zeile, in der Control-Break betätigt wird, wird nicht gespeichert. Nach Betätigung von Control-Break kehrt GW-BASIC zur Befehlsebene zurück ("OK").

Beispiel: AUTO 100,50

generiert Zeilennummern 100,150,200....

AUTO

generiert Zeilennummern 10,20,30,40....

Hinweis: Beachten Sie beim Editieren von Programmzeilen, daß, ausgenommen der gerade von AUTO angebotenen Zeile, AUTO nur durch Betätigung der Control-Break-Taste verlassen werden kann.

BEEP Befehl

Syntax: BEEP

Verwendung: Zur Erzeugung eines Tones von 830 Hz im Lautsprecher, für eine Dauer von ungefähr 1/4 Sekunde (genauer: 240 ms).

Bemerkung: BEEP hat die selbe Wirkung wie Print CHR$(7); (siehe Anhang B).

Beispiel:

```
2430 IF X < 20 THEN BEEP
```

BLOAD Befehl

Syntax: BLOAD <dateispez>[,<offset>]

<dateispez> ist die Bezugnahme auf eine Datei, in Übereinstimmung mit den NCR-DOS Dateibenennungsrichtlinien (siehe Kapitel 5).

<offset> ist ein numerischer Ausdruck im Bereich 0 bis 65535. Dieser stellt die Offset-Adresse dar, an welche die mit <dateispez> angegebene Datei geladen werden soll. Die Offset-Adresse bezieht sich auf das zuletzt von einem DEF SEG-Befehl definierte Programmsegment.

Verwendung: Laden einer angegebenen Speicherabbilddatei (memory image file) von Platte in den Speicher.

Bemerkungen: Der BLOAD-Befehl erlaubt es, ein Programm oder Daten, die als Speicherabbilddatei (memory image file) abgespeichert wurden, an jede beliebige Speicheradresse zu laden. Eine Speicherabbilddatei ist eine Byte-für-Byte-Kopie vom ursprünglichen Speicherinhalt.

Wird der Offset unterlassen, so werden die in der Datei enthaltene Segmentadresse und der zugehörige Offset (z.B. die von BSAVE bei der Dateieröffnung angegebene Adresse), benutzt. Daher wird die Datei in die gleiche Speicheradresse geladen, von der aus sie gespeichert wurde.

Wird der Offset angegeben, dann ist die verwendete Segmentadresse diejenige, welche im zuletzt ausgeführten DEF SEG definiert wurde. Wenn kein DEF SEG-Befehl gegeben wurde, wird das GW-BASIC-Datensegment als Standardadresse verwendet.

Achtung: BLOAD führt keine Adressenbereichsprüfung durch. Aus diesem Grund kann eine Datei an jede beliebige Stelle im Speicher geladen werden. Achten Sie darauf, daß GW-BASIC oder das Betriebssystem nicht überschrieben werden.

Beispiel:

```
10 DEF SEG=&H6000
20 BLOAD"PROG1",&HF000
```

Dieses Beispiel setzt die Segmentadresse auf hexadezimal 6000 und lädt PROG1, beginnend bei F000 Bytes (hexadezimal) über der Segmentadresse. (Sie sind nicht gezwungen hexadezimale Werte zu verwenden. Die entsprechenden Dezimalgegenwerte sind 24576 und 61440 respektive).

Hinweis:

BLOAD ist besonders geeignet, um Daten von der Platte in den Bildschirmspeicher zu laden.
Hierzu werden zusätzliche Informationen über die Anfangsadresse und den Aufbau des Bildschirmspeichers benötigt (siehe Kapitel 7).

BSAVE Command

Syntax: BSAVE <dateispez>,<offset>,<länge>

<dateispez> ist die Bezugnahme auf eine Datei, in Übereinstimmung mit den NCR-DOS Dateibenennungsrichtlinien (siehe Kapitel 5).

<offset> ist ein numerischer Ausdruck im Bereich 0 bis 65535. Dies ist die Offsetadresse, von dem zuletzt in einem DEF SEG-Befehl definierten Programmsegment. Der Speicherinhalt wird von dieser Stelle an auf Platte gespeichert.

<länge> ist ein numerischer Ausdruck im Bereich 1 bis 65535. Er stellt die Länge des Speicherbereichs in Bytes dar, der abgelegt werden soll.

Verwendung: Zur Abspeicherung des Inhalts eines angegebenen Speicherbereichs (z.B. ein Maschinensprachprogramm) als Plattendatei.

Bemerkung: Die Angaben für <dateispez>, <offset>, und <länge> werden in der Syntax des Befehls benötigt.

Der BSAVE-Befehl erlaubt die Speicherung von Daten oder Programmen als Speicherabbilddateien (memory image files) auf Platte. Eine Speicherabbilddatei (memory image file) ist die Byte-für-Byte-Kopie des Speicherinhalts.

Wird <offset> unterlassen, so wird die Fehlermeldung "unzulässiger Dateiname" (bad file name) ausgegeben, und das Speichern abgebrochen. Ein DEF SEG-Befehl muß vor einem BSAVE ausgeführt werden. Die zuletzt ermittelte DEF SEG-Adresse wird zum Sichern verwendet.

Wenn <länge> unterlassen wird, erscheint die Fehlermeldung "unzulässiger Dateiname" (bad file name) und die Speicherung wird abgebrochen.

Beispiel:

```
10 DEF SEG=$H6000
20 BSAVE"PROG1",&HF000.256
```

Dieses Beispiel speichert 256 Bytes, beginnend bei der Speicheradresse 6000:F000, das heißt, hexadezimal F000 Bytes über der Segmentadresse hexadezimal 6000, in der Datei PROG1. (Sie sind nicht gezwungen hexadezimale Werte zu verwenden. Die entsprechenden Dezimalgegenwerte sind respektive 24576 und 61440.)

Hinweis:

BLOAD ist besonders geeignet, um Daten von der Platte in den Bildschirmspeicher zu laden.
Hierzu werden zusätzliche Informationen über die Anfangsadresse und den Aufbau des Bildschirmspeichers benötigt (siehe Kapitel 7).

CALL Command

Syntax: CALL <variabler name>[(<argumente>)]

Wobei <variabler name> eine Adresse enthält, die den Ausgangspunkt der Unterroutine im Speicher darstellt. Dieser Ausgangspunkt ist eine Offset-Adresse, die sich auf das zuletzt definierte Segment bezieht. <variabler name> muß kein Feldvariablenname sein.

<argumente> enthält die Argumente, die an die externe Unterroutine weitergegeben werden. <argumente> darf nur Variable enthalten.

Verwendung: Zum Aufrufen einer Unterroutine in Assembler oder einer kompilierten Routine, die in einer anderen höheren Programmiersprache geschrieben wurde.

Bemerkungen: Der CALL-Befehl ist eine Möglichkeit den Ablauf des Programmes einer externen Unterroutine zu übergeben (siehe auch USR-Funktion).

Der CALL-Befehl erzeugt die gleiche Aufrufzeichenfolge (calling sequence), die von Microsoft FORTRAN und Microsoft BASIC Kompilern verwendet werden.

CDBL Funktion

Syntax: CDBL(X)

Verwendung: Zur Umwandlung von X in eine Zahl mit doppelter Genauigkeit

Beispiel:
```
10 A = 454.67
20 PRINT CDBL(A)
```
ergibt
454.6700134377344

Hinweis: Auch durch diese Funktion kann eine Zahl natürlich nicht genauer werden, als vor der Umwandlung. Im Beispiel ist die neue Zahl, mit doppelter Genauigkeit, immer noch bis zur zweiten Stelle nach dem Dezimalpunkt genau, jedoch nur nach Rundung. Der Abschnitt „Zahlentypumwandlung" in Kapitel 1 beschreibt die Faktoren, welche die Genauigkeit beim Umwandeln von einem Typ in einen anderen beeinflussen.

CHAIN Befehl

Syntax: CHAIN[MERGE]<programmname>[,[<zeilennummer>][,ALL][,DELETE<range>]]

Siehe die unten aufgeführten Beispiele für Erläuterungen der Syntaxoptionen

Verwendung: Zum Aufruf eines Programmes und zur Übergabe von Daten an dieses Programm aus dem laufenden Programm

Bemerkung: <programmname> ist die Bezeichnung des aufgerufenen Programmes (siehe Kapitel 5 „Dateien und Geräte".

Der COMMON-Befehl kann zur Übergabe von Variablen verwendet werden (siehe auch Beschreibung von COMMON).

<zeilennummer> ist eine Zeilennummer oder ein Ausdruck, der eine Zeilennummer ergibt, und bezieht sich auf das aufgerufene Programm. Sie ist der Ausgangspunkt für den Programmablauf des aufgerufenen Programmes. Wird die Zeilennummer nicht angegeben, so beginnt der Programmablauf mit der ersten Zeile. <zeilennummer> wird beim Editieren des aufrufenden Programmes nicht verändert.

Mit Hilfe der Angabe ALL werden alle Variablen im laufenden Programm an das aufgerufene Programm übergeben. Wird die ALL-Angabe unterlassen, muß der COMMON-Befehl eine Auflistung der zu übergebenden Variablen enthalten.

Wenn die All-Angabe verwendet wird, <zeilennummer> jedoch nicht, so muß ein Komma für <zeilennummer> stehen.
Zum Beispiel ist CHAIN "NEXTPROG", ALL richtig, CHAIN "NEXTPROG", ALL dagegen nicht. Im letzten Fall nimmt GW-BASIC an, daß ALL ein Variablenname ist und verwendet es als eine Zeilennummer.

Mit der Angabe MERGE kann ein Unterprogramm als Überlagerung in das GW-BASIC-Programm gebracht werden. Das heißt, das aufgerufene Programm wird in das ursprüngliche Programm eingebunden (siehe MERGE). Das so aufgerufene Programm muß im ASCII-Format gespeichert sein.

Nachdem eine Überlagerung verwendet wurde, ist es empfehlenswert sie wieder zu löschen, um Platz für eine neue Überlagerung zu schaffen. Dazu benützt man die DELETE-Angabe.

Die Zeilennummern in der Angabe <bereich> werden vom RENUM-Befehl beeinflußt.

Beispiele:

```
COMMON VAR1, VAR2, VAR $
CHAIN „NEWPROG“
```

veranlaßt GW-BASIC dieses Programm zu laden, und die Programmsteuerung an den Beginn des Programms zu übergeben. Die drei unter COMMON genannten Variablen werden dem mit CHAIN angehängten Programm zur Verfügung gestellt.

```
COMMON VAR1, VAR2, VAR$
CHAIN “NEWPROG”, 1000
```

hat die selbe Wirkung, mit der Ausnahme, daß die Programmsteuerung an die Zeile 1000 eines mit CHAIN angehängten Programmes übergeben wird.

```
CHAIN “NEWPROG”, 1000, ALL
```

unterscheidet sich vom vorherigen Beispiel dadurch, daß alle Variablen (nicht nur drei) des laufenden Programes dem mit CHAIN angehängten Programm zur Verfügung gestellt werden.

```
CHAIN MERGE “OVERLY1”, 1000, ALL
```

hat den besonderen Effekt, die Zeilen im laufenden Programm mit Zeilen von OVERLY1 da zu überschreiben, wo Zeilennummern zwischen den beiden Programmen übereinstimmen (OVERLY1 wird in der gesamten Länge angehängt).

Es ist möglich einen Bereich von Programmzeilen zu löschen, um genügend Speicherplatz für das mit CHAIN angehängte Programm zur Verfügung zu haben.

Zum Beispiel:

CHAIN MERGE "OVERLY2", 1000, ALL, DELETE 1000-2000

löscht die Zeilen 1000 bis 2000 des laufenden Programmes bevor OVERLY2 geladen wird.

Hinweis:

Der CHAIN-Befehl mit der MERGE-Angabe beläßt die Dateien geöffnet und behält den laufenden "OPTION BASE"-Status bei. Das mit CHAIN angehängte Programm kann jedoch über eine eigene OPTION BASE verfügen, wenn keine Feldvariablen übertragen werden.

Wird die MERGE-Angabe unterlassen, behält CHAIN die vorgegebenen Variablentypen oder Anwenderfunktionen im aufgerufenen Programm nicht bei. Das heißt, jeder DEFINT, DEFSNG, DEFDBL, DEFSTR oder DEFFN-Befehl, der eine gemeinsame Variable enthält, muß im angehängten Programm neu angegeben werden.

Wird die MERGE-Angabe verwendet, sollten die Anwenderfunktionen vor alle CHAIN MERGE-Befehle im Programm gestellt werden. Andernfalls bleiben die Anwenderfunktionen bis zur Ausführung dieser Funktionen unbelegt.

CHAIN führt vor dem Programmablauf einen RESTORE-Befehl aus. Deswegen wird der DATA-Zeiger zurückgesetzt. Der READ-Befehl setzt also nicht das Lesen der DATA-Werte wie im alten Programmm fort, sondern beginnt erneut.

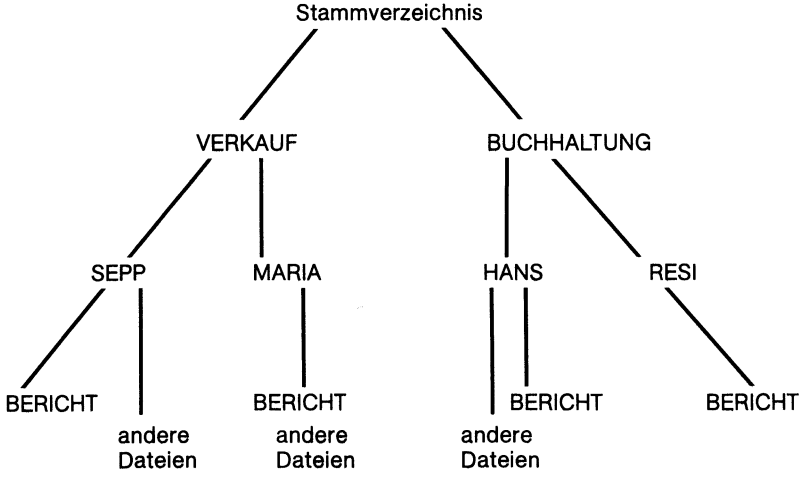
CHDIR Befehl

Syntax: CHDIR <pfad>

Verwendung: Änderung des aktuellen Inhaltsverzeichnisses

Bemerkung: <pfad> ist ein Zeichenkettenausdruck der 128 Zeichen nicht überschreitet, und bestimmt das neue Inhaltsverzeichnis, das jetzt das aktuelle Verzeichnis ist.

Beispiele: Wenn die folgende hierarchische Struktur für die Änderung zum Inhaltsverzeichnis SUE vom Inhaltsverzeichnis ROOT gegeben ist,



so verwenden Sie

CHDIR "BUCHHALTUNG/SUE"

Um von SALES zum Verzeichnis JOHN zu wechseln:

CHDIR "JOHN"

Um von JOHN zurück zu SALES zu wechseln wird

CHDIR "\"

verwendet.

Das Laufwerk kann gewechselt werden, wenn mit dem CHDIR-Befehl ein anderes als das gerade eingeschaltete Laufwerk angegeben wird. Zum Beispiel:

```
CHDIR "C:SALES"
```

Hinweis:

Wenn Ihr Programm auf eine Datei zugreift, sucht GW-BASIC im aktuellen Platten-Inhaltsverzeichnis nach dieser Datei. (Die Syntaxbeschreibungen in diesem Kapitel bezeichnen Dateizugriffe mit dem Ausdruck <dateispez>).

Hat Ihre Tastatur keine (\)-Taste, so können Sie auch die (/)-Taste zur Bezeichnung von Pfaden verwenden.

CHR$ Funktion

Syntax: CHR$(I)

Verwendung: Gibt ein Zeichen an, dessen ASCII-Zeichencodewert I ist, wobei I für eine Dezimalzahl von 0 bis 255 steht. (ASCII-Codes sind in Anhang B aufgeführt).

Bemerkung: CHR$ wird allgemein verwendet, um ein Sonderzeichen an den Bildschirm oder den Drucker zu senden. Zum Beispiel könnte ein akkustisches Zeichen (Piepton) CHR$(7) vor einer Fehlermeldung ausgegeben werden, oder ein Seitenvorschub (form feed) CHR$(12) könnte zum Löschen des Bildschirms und zum Zurückbringen der Schreibmarke in die Ausgangsposition abgeschickt werden.

Beispiele:

```
PRINT CHR$(66)
```

ergibt

```
B
```

Siehe die ASC-Funktion für Einzelheiten über die Umwandlung eines Zeichens zurück in seinen ASCII-Code. Siehe auch die Beispiele in der CONT-Beschreibung.

Mittels der STRING$-Funktion kann ein Wiederholungsfaktor für ein Zeichen gesetzt werden. Das folgende Beispiel läßt den Lautsprecher etwa 30 Sekunden lang ertönen:

```
10 B$=STRING$ (120,CHR$(7))
20 PRINT B$;
```

Das nächste Beispiel programmiert die Funktionstaste 1 die GW-BASIC-Kommandoliste auszugeben, ohne daß <CR> betätigt werden muß:

```
KEY 1, "LIST" + CHR$(13)
```

Hinweis: Die CHR$-Funktion ist hilfreich bei der Anzeige von Zeichen, für welche die Betätigung einer einzelnen Taste auf der Tastatur nicht möglich ist. Ihre Tastatur verfügt wahrscheinlich nicht über ein Quadratwurzelzeichen, es kann aber trotzdem mit

```
PRINT CHR$(251);
```

angezeigt werden.

CINT Funktion

Syntax: CINT(X)

Verwendung: Zur Umwandlung von X in eine Ganzzahl durch Rundung der Stellen nach dem Komma.

Bemerkung: Befindet sich das Ergebnis nicht im Bereich -32768 bis 32767, so tritt die Fehlermeldung „Überlauf" (overflow) auf.

Beispiel:

```
PRINT CINT(45.67)
ergibt
 46

PRINT CINT(-3.85)
ergibt
-4
```

Unter den Funktionen CDBL und CSNG finden Sie weitere Einzelheiten über die Umwandlung von Zahlen in einen Datentyp mit doppelter und einfacher Genauigkeit. Siehe auch die Funktionen FIX und INT , die beide Ganzzahlwerte angeben.

CIRCLE Befehl

Syntax: CIRCLE (x,y),<radius>[,<farbe>[,anfang, ende>[,<form>]]]

Verwendung: Zeichnet im Graphikmodus eine Ellipse auf den Bildschirm:

x,y

Bemerkung: Gibt die Koordinaten des Mittelpunkts der Ellipse an. Koordinaten können absolut oder relativ zum zuletzt adressierten Punkt am Bildschirm angegeben werden (unter Verwendung von STEP).

<radius>
Gibt den Radius (Hauptachse) in Punktwerten an.

<farbe>
Bei Farbgraphik in mittlerer und hoher Auflösung wird damit die Farbe (1 bis 3) der aktuellen Palette ausgewählt. Die Hintergrundfarbe (0) ist ebenfalls zulässig. Wenn keine Farbe angegeben wird, verwendet GW-BASIC die Farbe 3. In einfarbiger Graphik mittlerer und hoher Auflösung kann die Farbe 1 (für weiß) oder 0 (für schwarz) sein. Der Standardwert ist 1.

<anfang,ende>
Gibt den Winkel im Bogenmaß an, wo die Zeichnung beginnen oder enden soll. Die Werte können zwischen –2*PI bis 2*PI variieren, wobei PI dem Wert 3.141593 entspricht (Siehe auch „Bemerkungen").

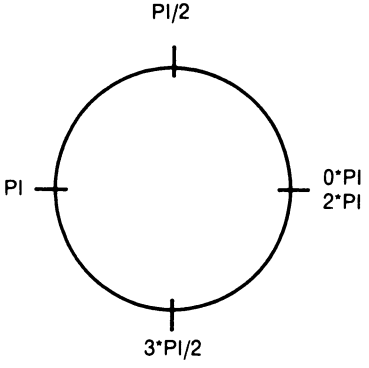
<form>
Gibt das Verhältnis zwischen dem x-Radius und dem y-Radius an. Ist das Verhältnis kleiner als 1, so ist der Radius der x-(Horizontal)-Radius. Wenn das Verhältnis größer als 1 ist, dann handelt es sich um den y-(Vertikal)-Radius.

GW-BASIC erzeugt einen Kreis auf dem Bildschirm, ohne daß vorher eine Angabe für den <form>-Wert gemacht wurde. Geben Sie für <form> einen Wert an, der nicht 5/6 in Graphik mit niedriger und hoher Auflösung oder 5/12 in Graphik

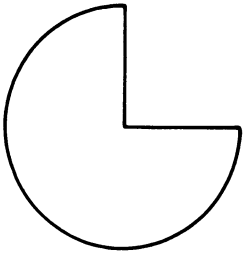
mit mittlerer Auflösung beträgt, wird eine Ellipse angezeigt.

Ist der Wert, den Sie angeben, kleiner als 5/6 (mittlere Auflösung: 5/12), so hat die Ellipse die Form eines in der Horizontalen gestreckten Achse (siehe Beispiel).

Die ersten beiden zu bearbeitenden Werte (arguments), die x- und y-Koordinaten und der Radius, sind die einzigen Werte, die zum Zeichnen eines Kreises benötigt werden. Die letzten beiden Argumente werden zum Zeichnen anderer „kreisartiger" Formen verwendet. „Anfang" und „Ende" ermöglichen es Ihnen, den Anfang und das Ende der gezeichneten Kreislinie selbst zu bestimmen. Die Werte von „Anfang" und „Ende" sind mathematisch korrekt im Bogenmaß anzugeben.



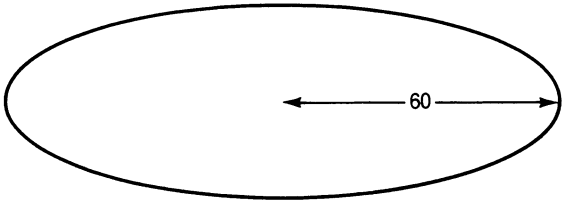
Die „Anfang"- und „Ende"-Werte können beide negativ sein (–0 ist jedoch nicht erlaubt), wobei der Winkel durch eine Linie mit dem Mittelpunkt verbunden ist. Die „Anfangs"- und „End"-Werte von –PI/2, –2PI würden zum Beispiel einen Teil eines Kreises zeichnen:



Verwenden Sie ein Form-Argumment, wenn Sie eine Ellipse zeichnen wollen. Wie Sie wissen ist r der x-Radius, wenn der Formfaktor kleiner als 1 ist. Ist dieser größer als 1, so ist r der y-Radius. Zum Beispiel:

```
10 SCREEN 1
20 CIRCLE (160,100),60,,,,5/18
```

zeichnet die folgende Ellipse



Hinweis:

Punkte die außerhalb des Bildschirms liegen werden nicht von CIRCLE gezeichnet, und rufen keine Fehlersituation hervor.

Es ist zulässig, daß die Mittelpunktkoordinaten außerhalb des Bildschirmbereiches liegen. Die Ellipse wird dann mit Hilfe eines gedachten Mittelpunktes gezeichnet, wobei Punkte, die innerhalb der eigentlichen Bildschirmkoordinaten liegen, angezeigt werden.

Das folgende Beispiel bewirkt die Darstellung eines Bogens in mittlerer Auflösung in der oberen rechten Ecke des Bildschirms:

Beispiel:

```
10 SCREEN 2
20 CIRCLE (650,-10),100
```

Sobald eine Ellipse gezeichnet ist, wird der zuletzt angesprochene Bildschirmpunkt von GW-BASIC als Mittelpunkt der Ellipse angenommen.

CLEAR Befehl

Syntax: CLEAR[,[<ausdruck1>][,<ausdruck2>]]

Verwendung: Setzt alle numerischen Variablen auf Null, löscht alle Zeichenvariablen, und schließt alle geöffneten Dateien. Wahlweise kann die obere Speichergrenze und der für den „Stack" (Stapelung) reservierte Speicherplatz gesetzt werden.

Bemerkung: <ausdruck1> ist eine Speicherstelle, die, falls angegeben, die höchste für den Arbeitsbereich von GW-BASIC zur Verfügung stehende Speicherposition definiert. Sie können so Platz für Maschinensprachprogramme reservieren.

<ausdruck2> reserviert Speicherplatz für den GW-BASIC-„Stack". Automatisch werden entweder 512 Bytes oder ein Achtel des verfügbaren Speichers (jeweils der kleinere Wert) angenommen. Die Spezifikation eines größeren „Stacks" kann erforderlich sein, wenn Ihr Programm tiefverschachtelte GOSUB-Routinen oder sehr viele FOR...NEXT-Schleifen verwendet, oder bei umfassender Verwendung des PAINT-Befehls.

CLEAR hat folgende Auswirkungen:

Schließen aller Dateien.
Löschen aller COMMON-Variablen.
Setzt numerische Variablen und Feldvariablen auf Null.
Setzt den „Stack" zurück und gibt den Speicherplatz frei.
Setzt alle Zeichenkettenvariablen und Feldvariablen auf Null.
Gibt alle Plattenpuffer frei.
Macht alle DEFinitionen ungültig (DEF FN, DEF USR, DEF SEG, DEFINT, DEFDBL, DEFSNG, DEFSTR).
Schaltet den Ton ab und führt zum Musikvordergrund zurück.
PEN und STRIG werden auf OFF zurückgesetzt.

Das im Speicher befindliche Programm wird von CLEAR nicht gelöscht.

Beispiele:

CLEAR

führt nur die obengenannten Aktionen durch.

CLEAR,32768

Setzt zusätzlich den maximalen Arbeitsbereich auf 32 KB.

CLEAR,2000

Reserviert 2000 Bytes für den "Stack".

CLEAR,32786,2000

Führt CLEAR durch, setzt den maximalen Arbeitsbereich auf 32 KB, und reserviert 2000 Bytes für den "Stack".

Hinweis:

Um Platz im Speicher zu schaffen, kann Ihr Programm für angegebene Feldvariablen den ERASE-Befehl verwenden.

CLOSE Befehl

Syntax: CLOSE[[#]<dateinummer>[,[#)<dateinummer...>]...]

Verwendung: Zum Abschließen der Ein-/Ausgabe an eine Datei, oder ein Gerät. Der CLOSE-Befehl hat die gegenteilige Wirkung des OPEN-Befehls.

Bemerkung: <dateinummer> ist die Nummer, unter welcher die Datei eröffnet wurde. Ein CLOSE-Befehl ohne zusätzliche Angaben schließt alle geöffneten Dateien und Geräte.

Die Beziehung zwischen einer bestimmten Datei und einer Dateinummer wird mit der Ausführung eines CLOSE-Befehls aufgehoben. Die Datei kann dann mit der selben oder einer anderen Dateinummer wiedereröffnet werden. Diese Dateinummer kann nun genauso zum Eröffnen einer beliebigen Datei verwendet werden.

Ein CLOSE-Befehl für eine Datei oder ein Gerät mit sequentieller Ausgabe schreibt zuerst den noch verbleibenden Pufferinhalt in die Datei.

Die Befehle END und NEW schließen immer alle Plattendateien automatisch. (STOP schließt Plattendateien nicht).

Zugriff auf Dateien und Geräte wird in Kapitel 5 beschrieben.

Beispiele: CLOSE

schließt alle geöffneten Dateien und Geräte.

CLOSE #1,#2,#3

schließt alle Dateien und Geräte denen die Nummern 1, 2 und 3 zugeordnet sind. (Die Benutzung des #-Zeichens steht Ihnen frei).

Hinweis: END, NEW, RESET, SYSTEM und RUN (ohne die R-Option) schließen geöffnete Dateien und

Geräte automatisch. Das Drücken der Tastenkombination „Control-Break" während des Programmablaufs bewirkt dasselbe. Mit dem STOP-Befehl können keine Dateien oder Geräte geschlossen werden.

CLS Befehl

Syntax: CLS<n>
wobei <n> für 1 oder 2 stehen kann

Bemerkungen: Ist der KEY-ON-Befehl in Kraft, während der CLS-Befehl benutzt wird, so wird der Bildschirm gelöscht; die Funktionszeile am unteren Ende des Bildschirms wird jedoch mit den gerade aktiven Vordergrund-/Hintergrundfarben erneuert.

Im Textmodus befindet sich der Cursor in der linken oberen Ecke am Bildschirm. Im Textmodus kann der Bildschirmspeicher 8 Bildschirmseiten speichern. CLS benützt nur die aktive Bildschirmseite (verwenden Sie SCREEN um festzustellen, welche Seite aktiv ist).

Im Graphikmodus ist nur eine Bildschirmseite im Bildschirmspeicher vorhanden. CLS löscht den Bildschirmspeicher gänzlich. Der zuletzt angesprochene Bildschirmpunkt wird dann als 160,100 in niedriger Auflösung, als 320,100 in mittlerer und als 320,200 in hoher Auflösung angenommen. Ein nachfolgender Graphikbefehl, unter Verwendung von STEP, bezieht sich auf diesen Punkt.

Durch Angabe des <n> Parameters können bestimmte Bildschirmteilabschnitte gelöscht werden. <n> kann für 1 oder 2 stehen. CLS 1 resultiert in der Löschung eines Anzeigeausschnittes, CLS 2 löscht den für die Textdarstellung vorgesehenen Bildschirmbereich.

Beispiel:
```
10 COLOR 12,1
20 CLS
```

Löscht im Textmodus den Bildschirm und stellt die Hintergrundfarbe auf blau (1).
Nachfolgender Text (einschließlich der GW-BASIC "Ok"-Meldung) erscheint in hellrot (12).

Hinweis: Im Textmodus stellt der COLOR-Befehl alleine den Bildschirm nicht auf die neue Hintergrundfarbe ein. Die neue Hintergrundfarbe wird erst beim Weiterschreiben verwendet. CLS stellt den gesam-

ten Textbereich des Bildschirms auf die Hintergrundfarbe um. Auch im Graphikmodus wird CLS nicht benötigt, um die Hintergrundfarbe der Bildschirmanzeige umzustellen.

Die Befehle SCREEN und WIDTH stellen den Bildschirmmodus und die Zeilenlänge ein und löschen zusätzlich den Bildschirm.

COLOR Befehl (Textmodus)

Syntax: COLOR [<schrift>][,[<hintergrund>][,<rahmen>]]

<schrift> ist ein numerischer Ausdruck im Bereich 0 bis 31. Dieser steht für die Farbe in welcher die Schrift angezeigt werden.

<hintergrund> ist ein numerischer Ausdruck im Bereich 0 bis 7 für die Hintergrundfarbe.

<rahmen> ist ein numerischer Ausdruck im Bereich 0 bis 15, der die Farbe des Rahmens außerhalb der Textwiedergabefläche festlegt. Beim NCR-Bildschirm ist dies aus technischen Gründen nicht möglich. Jedoch wirkt der Befehl auf entsprechende externe Bildschirme, die an den PC 4i angeschlossen werden können.

Verwendung: Zur Änderung eines oder mehrerer der zwei Farbfaktoren (schrift, hintergrund, rahmen), die den Aufbau der Anzeige auf einem Farbbildschirm bestimmen. Beim einfarbigen Bildschirm kann COLOR zum Umstellen verwendet werden (zum Beispiel schwarze Schrift auf weißem Hintergrund anstatt weiße Schrift auf schwarzem Hintergrund, oder umgekehrt). Bei beiden Bildschirmen (ein- und mehrfarbig) kann die Schrift heller oder blinkend dargestellt werden.

Bemerkungen: Die Bedeutung der Werte 0 bis 7 sind:

0 Schwarz	4 Rot
1 Blau	5 Magenta
2 Grün	6 Braun
3 Zyan	7 Weiß

Die Werte 8 bis 15 beziehen sich auf Versionen mit hoher Auflösung der selben Farben:

8 Grau	12 Hellrot
9 Hellblau	13 Weinrot
10 Hellgrün	14 Gelb
11 Türkis	15 Intensivweiß

Bei der Textfarbe können Sie auch zur gewünschten Farbkennziffer 16 addieren, und erhalten so

einen Wert zwischen 16 und 31. Dies bewirkt bei nachfolgenden Texteinträgen eine blinkende Anzeige in der gewählten Farbe.

Der Adapter mit einfarbigem Bildschirm verwendet die folgenden Farbwerte (unter der Farbe weiß versteht man hier immer die Standardschriftfarbe, die Ihr Bildschirm verwendet).

	Schrift	Hintergrund
0	Schwarz	Schwarz
1	Weiß, und unterstrichen	Schwarz
2...6	Weiß	Schwarz
7	Weiß	Weiß

Bei Verwendung eines weißen Hintergrundes (7) können Sie zum Schreiben 0,8,16 oder 24 benutzen (die beiden letzten Werte erzeugen eine blinkende Anzeige). Sie können jedoch nicht weiße Schrift auf weißem Hintergrund einstellen. Außerdem kann die Schrift bei weißem Hintergrund nicht auf normale Helligkeit und Unterstreichung eingestellt werden.

Wenn Sie die Hintergrundfarbe schwarz (0...6) verwenden, können Sie für die Schrift weiß, weiß halbhell, weiß blinkend und weiß halbhell blinkend wählen (der Reihenfolge nach 7,15,23 und 31). Wollen Sie unterstrichene Zeichen mit diesen Eigenschaften erzeugen, so ist von der jeweiligen Kennzahl die Zahl 6 abzuziehen.

Am sinnvollsten ist es natürlich, die Farbzusammenstellung so zu wählen, daß ein vernünftiger Kontrast zwischen Hintergrund und Text entsteht. GW-BASIC hindert Sie jedoch nicht daran, zweimal schwarz zu verwenden. Um diesen unsichtbaren Schrifteffekt zu erzeugen, brauchen Sie nur 0 für den Hintergrund und 0, 8, 16 oder 24 für den Text einzutragen. Sie könnten diese Möglichkeit nutzen, um zum Beispiel Kennworte, die nicht am Bildschirm erscheinen sollen, einzugeben.

Andere als die hier erwähnten Kombinationen führen zu weißer Schrift auf schwarzem Hintergrund.

Bei einem Adapter mit Farbbildschirm können Sie, um den Text unsichtbar zu machen, jede Farbkennzahl zwischen 0 und 7 gemeinsam für <schrift> und <hintergrund> verwenden.

Beispiele: (für einen Adapter mit Farbbildschirm)

COLOR 12

stellt für den Text die Farbe hellrot ein. Die Hintergrundfarbe bleibt unverändert.

COLOR ,1

ändert nur die Hintergrundfarbe; die neue Farbe ist blau.

COLOR 12,1

setzt die Schrift auf hellrot, und den Hintergrund auf blau.

Das folgende Beispiel gilt für den einfarbigen Bildschirm sowohl als auch für den Farbbildschirm. Dabei ist es erforderlich einen bekannten Text so einzugeben, daß das Geschriebene nicht am Bildschirm zu sehen ist. Sie können jedoch mit der Backspace-Taste zürückgehen und eventuelle Fehler korrigieren. Ihre Eingabezeit wird gemessen und in Sekunden angezeigt, wobei angenommen wird, daß der eingegebene und mit der <CR>- Taste abgeschickte Text mit der in X$ enthaltenen Zeichenkette identisch ist.

Nachdem Ihr Ergebnis angezeigt wurde, können Sie durch Drücken einer beliebigen Taste zu "Ok" zurückkehren.

```
10 X$="Wenn der Hund mit der Wurst über den
   Eckstein springt"
20 SCREEN 0:WIDTH 80
30 COLOR 7,0
40 CLS
50 PRINT "Geben Sie ein...";
60 FOR DLY%=1 TO 700:NEXT DLY%
70 COLOR 0,7
80 PRINT "JETZT"
```

```
90 TIME$="00:00:00"
100 COLOR 0,0
110 INPUT I$
120 T$=RIGHT$(TIME$,2)
130 SLOW$=RIGHT$(TIME$,4)
140 CLS
150 IF I$()X$ THEN GOTO 220
160 COLOR 23,0
170 PRINT X$:PRINT:PRINT
180 PRINT " GUT GEMACHT"
190 IF ASC(SLOW$)<>48 THEN PRINT "Sie
    haben aber mindestens eine Minute
    gebraucht": GOTO 260
200 PRINT "Sie brauchten ";T$" Sekunden":-
    PRINT
210 GOTO 260
220 COLOR 7,0
230 PRINT "Nicht ganz richtig...":PRINT
240 PRINT "Sie hätten eingeben sollen:":
    PRINT X$:PRINT
250 PRINT „...das haben Sie geschrieben:":
    PRINT I$
260 COLOR 7,0
270 IF INKEY$="" THEN GOTO 270
280 CLS
290 END
```

Hinweis:

Die abschließende Angabe in einem COLOR-Befehl sollte kein Komma sein.

Ein Wert der sich außerhalb des erlaubten Bereichs befindet, kann zu einem unerlaubten Funktionsaufruf ("illegal function call" error) führen.

Die Art der Anzeige (Textmodus, niedrige, mittlere oder hohe Graphikauflösung) kann durch den SCREEN-Befehl bestimmt werden.

COLOR Befehl (Graphikmodi)

Syntax: COLOR [<hintergrund>][,[<palette>]

<hintergrund> ist ein numerischer Ausdruck der die Hintergrundfarbe des Bildschirms angibt. Werte zwischen 0 und 15 sind zulässig. (Siehe COLOR (Textmodus) hinsichtlich der zu diesen Werten gehörenden Farben).

<palette> ist ein numerischer Ausdruck mit dem eine der beiden zur Verfügung stehenden Farbpaletten ausgewählt wird. Eine gerade Zahl wählt Palette 0, eine ungerade Zahl Palette 1.
Auf jeder Palette befinden sich drei Farben:

Farbnummer	Palette 0	Palette 1
1	Grün	Zyan
2	Rot	Magenta
3	Braun	Weiß

Verwendung: Bei Verwendung des entsprechenden COLOR-Befehls wird die neue Hintergrundfarbe sofort angezeigt. Wird mit dem COLOR-Befehl die Palette gewechselt, ändern sich die Farben der momentan dargestellten Zeichnungen entsprechend. Wenn Sie von Palette 0 auf Palette 1 wechseln, wird aus grün zyan, rot wird zu magenta, und was braun war wird weiß. Der umgekehrte Fall tritt ein, wenn Sie von Palette 1 zu Palette 0 wechseln (zyan wird grün, usw.)

Die Befehle CIRCLE, DRAW, LINE, PAINT, PRESET und PSET können entweder die Hintergrundfarbe oder eine der drei Farben aus der aktuellen Farbpalette verwenden. COLOR dient zur Auswahl einer Palette für diese Graphikbefehle.

Bemerkungen: Zeichen auf dem Bildschirm in Farbgraphik mit niedriger oder hoher Auflösung verwenden Farbnummer 3 aus der gegenwärtig ausgewählten Palette, das heißt, braun oder weiß.

Beispiele:

```
10 SCREEN 4
20 COLOR 2,0
```

setzt den Bildschirmmodus 4 (Farbgraphik mit hoher Auflösung), stellt den Hintergrund auf grün, und wählt die Farbpalette 0.

```
10 SCREEN 4
20 COLOR 4
```

setzt den Bildschirmmodus 4 und stellt die Hintergrundfarbe auf rot um. Die Farbpalette so wie vorher.

```
10 SCREEN 4
20 COLOR ,1
```

setzt den Bildschirmmodus 4 und wählt die Farbpalette 1. Die Hintergrundfarbe ändert sich nicht.

Das Beispiel am Ende des Kapitels „Die Bildschirmanzeige" zeigt nur einige der Möglichkeiten der GW-BASIC-Farbgraphik.

Hinweis:

COLOR ist bei einfarbiger Graphik mit mittlerer oder hoher Auflösung nicht anwendbar, da dieser Bildschirmmodus nur schwarz und weiß verwendet. Der Versuch den COLOR-Befehl in diesem Modus zu benutzen resultiert in einem unzulässigen Funktionsaufruf ("illegal function call error"). Diese Fehlermeldung tritt auch bei einem Wert über 255 auf.

COM Befehl

Syntax:

COM(n) ON
COM(n) OFF
COM(n) STOP

wobei n die Nummer für den Kommunikationskanal ist, diese ist entweder 1 oder 2.

Verwendung:

Freigeben oder Sperren der Unterbrechungsroutinen für Kommunikationsvorgänge auf dem angegebenen Kanal.

COM(n) ON gibt Unterbrechungsroutinen für Kommunikation durch einen ON COM-Befehl frei (siehe ON COM). Sind diese Routinen freigegeben und ist eine von Null abweichende Zeilennummer im ON COM-Befehl genannt, prüft GW-BASIC vor der Ausführung jedes Befehls, ob ein Signal auf der Kommunikationsleitung anliegt. GW-BASIC übergibt die Programmsteuerung an die vom ON COM-Befehl angezeigte Zeile.

COM(n) OFF sperrt Unterbrechungsroutinen für Kommunikation. Tritt ein Vorfall ein, wird er gespeichert und ON COM führt, sobald die Unterbrechung freigegeben ist, eine Übergabe der Programmsteuerung durch.

Beispiel:

```
10 COM(1) ON
```

Gibt Fehlerunterbrechungsroutine (error trapping) für Kommunikationskanal 1 frei.

COMMON Befehl

Syntax: COMMON <liste der variablen>

Verwendung: Zur Übergabe von Variablen an ein Programm, das mit CHAIN verkettet wird.

Bemerkung: COMMON wird im Zusammenhang mit CHAIN benutzt. COMMON-Befehle können an beliebigen Stellen im Programm erscheinen. Es wird jedoch empfohlen, Sie an den Beginn des Programmes zu stellen. Die selbe Variable kann nicht in mehr als einem COMMON-Befehl erscheinen. Feldvariablen werden gekennzeichnet durch das Anhängen von Klammern „()" an den Variablennamen. Sollen alle Variablen übergeben werden, empfiehlt es sich, CHAIN zusammen mit der All-Option zu verwenden und COMMON nicht zu verwenden.

CHAIN sollte nicht die Dimensionen einer Feldvariablen angeben. Ein derartiger Befehl wird von GW-BASIC ignoriert.

Beispiele:

```
100 COMMON A,B,C,D(),G$
110 CHAIN "PROG3",10
```

hängt das Programm PROG3 an, stellt die Variablen A,B,C,G$ und die Feldvariable D zu dessen Verfügung und beginnt mit dem Ablaufen des angehängten Programmes bei Zeile 10.

CONT Befehl

Syntax: CONT

Verwendung: Zur Fortsetzung des Programmablaufes nach Drükken der Taste Control-Break, der Taste STOP oder END, oder nach Auftreten eines Fehlers ohne Unterbrechungsroutine (untrapped error).

Bemerkung: Das Programm wird dort fortgesetzt wo es unterbrochen wurde. Falls die Unterbrechung an einer Stelle stattfand, an der die Aufforderung zur Dateneingabe innerhalb eines INPUT-Befehls erschien, wird das Programm mit dem erneuten Anzeigen dieser Aufforderung (? oder Text) fortgesetzt.

CONT wird normalerweise zusammen mit STOP zum Testen von Programmen verwendet. Nach der Programmunterbrechung können Zwischenwerte von Variablen mit Befehlen im Direktmodus überprüft und geändert werden. Die Programmausführung wird mit CONT oder einem GOTO im Direktmodus fortgesetzt. Mit GOTO kann die Fortsetzung bei einer angegebenen Zeilennummer aufgenommen werden. CONT kann zur Programmfortsetzung verwendet werden, nachdem ein Fehler ohne Unterbrechungsroutine GW-BASIC veranlaßt hat, den Programmablauf zu unterbrechen.

CONT ist nicht möglich, wenn das Programm während der Unterbrechung verändert wurde.

Beispiel: Das folgende Programm stellt die Zeichen des GW-BASIC Zeichensatzes dar, die einen Codewert von 128 oder höher haben (siehe Anhang B). Jedes Zeichen wird zusammen mit dem entsprechenden Codewert in einer eigenen Zeile ausgegeben. Eine Verzögerung beim Bildschirmrollen ist eingebaut (Zeile 50), Sie werden aber zur Untersuchung eines Zeichens für die Dauer eines beliebigen Zeitraums die Control-Break-Taste drücken müssen. Wenn Sie CONT als direkten Befehl eingeben, wird mit der Textanzeige fortgefahren. Sie können das Programm beliebig oft unterbrechen und wieder fortsetzen. WIDTH 40 (die Breite 40) sagt aus, daß die Zeichen in Großformat dargestellt werden.

```
10 WIDTH 40
20 FOR LOOP% = 128 TO 255
30 PRINT "Das Zeichen für Code";
   LOOP%;"ist ";CHR$(LOOP%) 40 PRINT
50 FOR DLY%= 1 TO 500:NEXT DLY%
60 NEXT LOOP%
```

Hinweis:

Der Befehl RUN eignet sich nicht zur Fortsetzung eines Programmes nach einer Unterbrechung, da er für den Speicherinhalt den selben Effekt hat wie CLEAR, nämlich die Schließung aller Dateien, Aufhebung aller DEFinitionen und Nullsetzung aller Variablen.

COS Funktion

Syntax: COS(X)

Verwendung: Ergibt den Kosinus von X in Bogengraden.

Bemerkung: Die Berechnung des COS(X) erfolgt mit einfacher Genauigkeit.

Beispiel:
```
10 X=2*COS(.4)
20 PRINT X
```
ergibt
```
 1.842122
```

Hinweis: Zur Umwandlung von Bogenmaß in Winkelgrade:

DEGREES=RADIANS*180/PI

wobei PI (einfache Genauigkeit) 3.141593 ist

Zur Umwandlung von Winkelgraden in Bogenmaß:

RADIANS=DEGREES*PI/180

CSNG Funktion

Syntax: CSNG(X)

Verwendung: Wandelt X in eine Zahl mit einfacher Genauigkeit um.

```
10 A#=482.342122
20 PRINT CSNG(A#)
```
ergibt
482.3421

Siehe die Funktionen CINT und CDBL zur Umwandlung in Ganzzahlwerte und Zahlen mit doppelter Genauigkeit.

Der Abschnitt „Zeichenumwandlung" in Kapitel 1 gibt weitere Informationen über Umwandlungsgenauigkeit.

CSRLIN Funktion

Syntax: CSRLIN

Verwendung: Variable, welche die aktuelle Zeilenposition des Cursors enthält (0 bis 24).

Beispiel: Im folgenden Beispiel wird von Zeile 10 die aktuelle Zeilenposition in L% eingelesen. Zeile 20 liest die aktuelle Spaltenposition in C% ein. Zeile 30 zeigt HALLO in der mittleren Zeile des Bildschirms an und Zeile 40 setzt die Position der Schreibmarke auf die vorherige Zeile und Spalte zurück.

```
10 L%= CSRLIN
20 C%= POS(0)
30 LOCATE 13,30:PRINT"HALLO"
40 LOCATE L%,C%
```

Hinweis: Es besteht eine Ähnlichkeit zwischen CSRLIN und POS, welches aber die aktuelle Spalte des Cursors angibt. LOCATE wird zur Positionierung des Cursors verwendet.

CVI, CVS, CVD
Funktion

Syntax:

CVI(<2-Byte-Text)
CVS(<4-Byte-Text)
CVD(<8-Byte-Text)

Verwendung:

Zur Umwandlung von Textwerten in numerische Werte.

Bemerkung:

Numerische Werte, die von einer Plattendatei mit wahlfreiem Zugriff gelesen wurden, müssen von Text zurück in Zahlen gewandelt werden. CVI wechselt einen 2-stelligen Text in einen Ganzzahlwert um. CVS wandelt einen 4-stelligen Text in eine Zahl mit einfacher Genauigkeit, und CVD wandelt einen 8-stelligen Text in eine Zahl mit doppelter Genauigkeit. In jedem einzelnen Fall wird das Ergebnis in der numerischen Variablen gespeichert, der Text selbst wird von der Umwandlung nicht beeinflußt.

Beispiel:

```
70 FIELD #1,4 AS N$, 12 AS B$
80 GET #1
90 Y=CVS(N$)
```

Der Datensatz, der von der Datei mit wahlfreiem Zugriff in Zeile 80 gelesen wurde, wird von der Anweisung FIELD von Zeile 70 in zwei Zeichenkettenvariablen N$ und B$ aufgeteilt. Zeile 90 betrachtet N$ als das Zeichenkettenäquivalent einer Zahl mit einfacher Genauigkeit, und überträgt den entsprechenden numerischen Wert auf Y. Wahrscheinlich war N$ ursprünglich eine Zahl, die mit Hilfe der MKS$-Funktion auf die Datei geschrieben wurde.

Hinweis:

Die Funktionen MKI$, MKS$ und MKD$ führen die umgekehrten Vorgänge durch, das heißt, sie wandeln numerische Werte in Zeichenketten um.

DATA Befehl

Syntax: DATA <konstante>[,<konstante>]...

Verwendung: Zum Speichern von numerischen und alphanumerischen (string-) Konstanten, die durch den READ-Befehl eines Programmes abgefragt werden.

Bemerkung: DATA weist GW-BASIC nicht direkt an, eine Anweisung auszuführen, und kann deshalb an beliebiger Position innerhalb eines Programmes stehen. Eine DATA-Liste kann so viele Konstanten enthalten, wie in eine Programmzeile passen. Es können beliebig viele DATA-Zeilen in einem Programm vorkommen. READ-Befehle greifen auf DATA-Zeilen in der Reihenfolge der Zeilennummern zu. Die Daten, die in den verschiedenen DATA-Zeilen enthalten sind, kann man sich als eine einzige ununterbrochene Reihe vorstellen. Unabhängig davon ist, wie viele Konstanten in einer Zeile stehen, und wo diese Zeile innerhalb des Programmes plaziert ist.

Eine DATA-Zeile kann numerische Konstanten aller Formate enthalten (d.h. Festkomma-, Gleitkomma-, Ganzzahl-, Dezimal-, Oktal- oder Hexidezimal-Konstanten). Dabei sind numerische Ausdrücke nicht erlaubt. String-(Zeichenketten)-Konstanten in DATA-Listen müssen nur dann in Anführungszeichen stehen, wenn sie Kommas, Doppelpunkte oder vorausgehende bzw. nachfolgende Leerzeichen enthalten. Andernfals können die Anführungszeichen entfallen.

Die Art der Variablen (numerisch oder alphanumerisch) im READ-Befehl muß mit der entsprechenden Konstanten in der DATA-Liste übereinstimmen. Andernfalls wird von GW-BASIC ein Syntaxfehler (keine Fehleingabe "type mismatch") erkannt.

Eine DATA-Zeile kann mit dem RESTORE-Befehl wieder von vorne gelesen werden.

Beispiel: Siehe READ.

DATE$ Befehl

Syntax: DATE$=<zeichenkettenausdruck>

<zeichenkettenausdruck> gibt eine Zeichenkette in einer der folgenden Formen aus:

mm-dd-yy
mm-dd-yyyy
mm/dd/yy
mm/dd/yyyy

Verwendung: Zum Einstellen des aktuellen Datums. Dies bewirkt das Gegenteil der DATE$-Funktion, die das aktuelle Datum wiederherstellt.

Bemerkung: Die Jahreszahl soll im Bereich 1980 bis 2099 eingegeben werden. Wenn der <zeichenkettenausdruck> nur eine Ziffer für den Tag oder Monat enthält, nimmt GW-BASIC eine 0 davor an. Wenn nur zwei Ziffern für das Jahr angegeben werden, nimmt GW-BASIC an, daß dieses in das 20. Jahrhundert fällt und setzt "19" davor.

Hinweis: Das Datum kann von NCR-DOS vor dem Laden von GW-BASIC bereits gesetzt worden sein. Dies hindert Sie jedoch nicht daran, es mit GW-BASIC einzustellen.

DATE$ Funktion

Syntax: DATE$

Verwendung: Eine Variable, die das aktuelle Datum enthält. (Verwenden Sie zum Einstellen des Datums den DATE$-Befehl.)

Bemerkung: Die DATE$-Funktion zeigt eine 10-stellige Zeichenkette in der Form mm-dd-yyyy an, wobei mm für Monat steht (o1 bis 12), dd für den Tag (01 bis 31), und yyyy für das Jahr (1980 bis 2099).

Beispiel: Wenn

```
10 PRINT DATE$
```

auf die Einstellung von Daten in der DATE$-Befehlbeschreibung folgt, wird

```
07-13-1984
```

angezeigt.

Die Trennungszeichen sind Bindestriche, auch wenn die Trennungszeichen beim Eingeben des Datums Schrägstriche waren.

DEF FN Befehl

Syntax: DEF FN <name>[(parameterliste>)]=
<funktionsdefinition>

Verwendung: Definition und Benennung einer Funktion, zusätzlich zu den von GW-BASIC bereitgestellten Funktionen.

Bemerkung: <name> muß ein gültiger Variablenname sein. Dieser Name wird, unter Voranstellung von FN, der Name der Funktion.

<parameterliste> besteht aus den Variablen in einer Funktionsdefinition, für die beim Aufrufen der Funktion Werte eingesetzt werden. Die Parameter werden durch Kommas getrennt.

<funktionsdefinition> ist die Einzeloperation, die von der Funktion ausgeführt wird. Variablennamen, die in diesem Ausdruck auftreten, dienen nur zur Definition der Funktion. Sie beeinflussen nicht Programmvariablen mit dem selben Namen. Ein Variablenname in einer Funktionsdefinition kann wahlweise in der Parameterliste erscheinen. Wenn dies der Fall ist, so wird der Wert der Variablen beim Aufruf der Funktion eingesetzt, andernfalls wird der aktuelle Wert der Variablen verwendet.

Die Variablen in der Parameterliste stellen auf der Basis eins zu eins die Argumente (Variablen oder Werte) dar, die beim Aufruf der Funktion einzusetzen sind.

DEF FN kann numerische oder auch alphanumerische Funktionen definieren. Ist die Funktion numerisch, so wird das Ergebnis des Ausdruckes der die Funktion enthält, an den aufrufenden Befehl mit der im Funktionsnamen enthaltenen Genauigkeit ausgegeben. Wenn dieser Befehl versucht, ein numerisches Funktionsergebnis an eine Zeichenkettenvariable zu übertragen, oder umgekehrt, tritt ein Eingabefehler (type mismatch) auf. Bevor eine neue Funktion von GW-BASIC verwen-

det wird, muß sie mit dem entsprechenden DEF FN-Befehl definiert werden, andernfalls erscheint die Fehlermeldung "undefined user function" (Benutzerfunktion nicht definiert). Eine Funktion kann mehr als einmal definiert werden. GW-BASIC bezieht sich immer auf die zuletzt angegebene Zuordnung.

Beispiel:

Zur Berechnung der Hypotenuse eines rechtwinkligen Dreiecks (die gegenüberliegende Seite des rechten Winkels) kann die Funktion wie folgt definiert werden:

```
10 DEF FNHYPOT (S1,S2)=SQR(S1^2+S2^2)
```

Um diese Funktion zu verwenden, könnten Sie mit dem Programm fortfahren mit

```
20 INPUT "Liegt eine Seite am rechten Winkel
   an?"
30 INPUT "Die andere Seite" SIDE 2
40 PRINT "Die Länge der Hypotenuse ist
   ";FNHYPOT (SIDE1,SIDE2)
```

Hinweis:

GW-BASIC nimmt DEF FN im direkten Modus nicht an.

Eine Funktion kann rekursiv sein, das heißt, sie kann sich selbst aufrufen. Sie müssen dann jedoch einen Weg finden sie wieder anzuhalten, da sonst eine Fehlersituation eintritt (out of memory – "kein Speicherplatz mehr vorhanden").

DEFINT/SNG/DBL/STR-Befehle

Syntax: DEF<Typ> <Bereich(e) von Buchstaben>

<Typ> entspricht INT, SNG, DBL oder STR.

Verwendung: Diese Befehle werden dazu benutzt, Variablentypen als Ganzzahlen, als Variablen mit einfacher Genauigkeit, Variablen mit doppelter Genauigkeit oder Zeichenfolgen zu deklarieren.

Bemerkungen: Die Variablennamen, die mit den Buchstaben beginnen, die in <Bereich von Buchstaben> angegeben werden, weisen den in <Typ> angegebenen Variablentyp auf. Jedoch ist ein Zeichen für die Typendeklaration am Ende des tatsächlichen Namens der Variablen (%, !, # oder $) stets gegenüber DEF-Typ vorrangig. (Siehe den Abschnitt "Variablen" in Kapitel 1.)

Werden keine Befehle zur Typendeklaration angetroffen, so geht GW-BASIC davon aus, daß es sich bei sämtlichen Variablen ohne Deklarationszeichen um Variablen mit einfacher Genauigkeit handelt.

Beispiele: 10 DEFDBL L-P

Sämtliche Variablen, die mit den Buchstaben L, M, N, O und P beginnen, sind Variablen mit doppelter Genauigkeit.

10 DEFSTR A

Sämtliche Variablen, die mit dem Buchstaben A be-ginnen, sind Zeichenfolgenvariablen. 10 DEFINT I-N,W-Z

Sämtliche Variablen, die mit den Buchstaben I, J, K, L, M, N, W, X, Y, Z beginnen, sind ganzzahlige Variablen.

Hinweis: GW-BASIC erkennt die in DEF-Typ festgelegte Typendeklaration erst, wenn der DEF-Typ-Befehl tatsächlich während der Programmausführung angetroffen wird. Deshalb sollten diese Definitionen mit DEF am Anfang des Programms stehen.

DEF SEG-Befehl

Syntax: DEF SEG [<=Adresse>]

Wobei "Adresse" ein numerischer Ausdruck in dem Bereich von 0 bis 65535 ist.

Verwendung: Die angegebene Adresse wird gesichert, damit sie als das für BLOAD, BSAVE, CALL, POKE, USE und PEEK erforderliche Segment benutzt werden kann.

Bemerkungen: Die Eingabe eines beliebigen Wertes außerhalb des "Adressen"-Bereichs von 0 bis 65535 führt zu der Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf). Der vorhergehende Wert wird in diesem Fall beibehalten.

Wird die Option "Adresse" weggelassen, so wird das zu benutzende Segment auf das GW-BASIC Datensegment (DS) gesetzt. Hier handelt es sich um den ursprünglichen Standardwert.

Wird die Option "Adresse" angegeben, so muß sie auf eine 16-Byte-Grenze ausgerichtet sein. GW-BASIC multipliziert diesen Wert mit 16 und benutzt das Ergebnis dieser Multiplikation als die tatsächliche Speicheradresse für den Segmentanfang. GW-BASIC überprüft die Gültigkeit der angegebenen Adresse nicht.

Beispiel: 10 DEF SEG=&HB800

Mit dieser Programmzeile wird das Segment auf die Hexadezimalzahl B800 (in Dezimalform: 47104) gesetzt. Diese Zahl stellt eine echte Speicheradresse dar, die 16 mal dem Wert von -B8000 (753664) entspricht. In der Tat handelt es sich hier um den Anfang des Bildschirmpuffers für den Farbbildschirm, so daß auf diesen Befehl wahrscheinlich BLOAD oder BSAVE folgen wird. Im weiteren Verlauf des Programms werden Sie das Segment wahrscheinlich wieder auf das GW-BASIC Datensegment ändern.

Hinweis: DEF und SEG müssen durch ein Leerzeichen voneinander getrennt werden. Ansonsten interpretiert GW-BASIC DEFSEG=100 so, als müsse der Wert 100 der Variablen DEFSEG zugewiesen werden.

DEF USR-Befehl

Syntax: DEF USR [<Ziffer>] =<ganzzahliger Ausdruck>

Verwendung: Gibt die Anfangsadresse einer Subroutine in der Assemblersprache an.

Bemerkungen: "Ziffer" kann eine beliebige Ziffer von 0 bis 9 sein. Die Ziffer entspricht der Nummer der USR-Subroutine, deren Adresse angegeben wird. Wird "Ziffer" weggelassen, so wird von DEF USR0 ausgegangen. Der Wert des „ganzzahligen Ausdrucks" ist die Startadresse des Relativzeigers der USR-Subroutine zu dem Segmentwert, der benutzt wird, wenn die USR-Subroutine aufgerufen wird.

Dieselbe "Ziffer" kann in mehr als einem DEF USR Befehl benutzt werden, wobei dieser Ziffer dann eine neue Adresse zugewiesen wird. GW-BASIC erkennt stets die als letztes zugewiesene Adresse. Dadurch kann auf mehr als zehn Subroutinen zugegriffen werden.

Beispiel:

```
200 DEF USR0=24000
210 X=USR0(Y^2/2.89)
```

Zeile 200 definiert die Startadresse einer in der Maschinensprache geschriebenen Subroutine als 24000. Zeile 210 weist der Variablen X das Ergebnis der Funktion zu, die die Subroutine mit dem Wert des in Klammern angegebenen Ausdrucks ausführt (siehe USR).

Muß auf eine Subroutine über eine absolute Speicheradresse zugegriffen werden, so betrachten Sie bitte folgendes Beispiel:

```
200 DEF SEG = 0
210 DEF USR0 = ABSADDR%
.
.
.
300 RESULT = USR0(INFO)
```

wobei ABSADDR% die absolute Speicheradresse der Subroutine enthält, auf die in Zeile 300 zugegriffen wird.

DELETE-Befehl

Syntax:

DELETE [<Zeilennummer1>]
[-<Zeilennummer2>]
DELETE [<Zeilennummer1>-]

Verwendung:

Löschen von Programmzeilen.

Bemerkungen:

GW-BASIC kehrt stets auf Befehlsebene ("Ok") zurück, nachdem ein DELETE-Befehl ausgeführt wurde. Ist eine angegebene Zeilennummer nicht vorhanden, so kommt es zu der Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

Beispiele:

DELETE 40
Löscht Zeile 40.

DELETE 40-100
Löscht die Zeilen 40 bis 100 einschließlich.

DELETE -40
Löscht sämtliche Zeilen bis einschließlich Zeile 40.

DELETE 40-
Löscht Zeile 40 und die nachfolgenden Zeilen in dem Programm.

DELETE
Löscht die aktuelle Zeile.

DIM Befehl

Syntax: DIM Variable (Indizes)
[,Variable(Indizes)] ...

Verwendung: Gibt die Höchstwerte der Indizes von Matrixvariablen an und weist den Speicherplatz dementsprechend zu.

Bemerkungen: Wird der Name einer Matrixvariablen ohne einen DIM-Befehl benutzt, so wird davon ausgegangen, daß der Höchstwert der Indizes der Matrix 10 beträgt. Wird ein Index benutzt, der größer ist als der angegebene Höchstwert, so kommt es zu einer Fehlermeldung "Subscript out of range" (Index außerhalb des Bereichs). Der Mindestwert für einen Index lautet stets 0, es sei denn, mit OPTION BASE wird ein anderer Wert angegeben.

Durch den DIM-Befehl werden sämtliche Elemente der angegebenen numerischen Matrizes auf einen Ausgangswert von Null gesetzt. Die Elemente einer Zeichenfolgenmatrix sind ursprünglich leer (haben eine Länge von Null).

In einem DIM-Befehl sind maximal 255 Dimensionen zulässig, ein in der Praxis kaum durchführbarer Grenzwert. Die Anzahl von Dimensionen sollte eher durch den zur Verfügung stehenden Speicherplatz und durch die maximal zulässige Länge für eine Programmzeile begrenzt werden. Die Höchstzahl von Elementen pro Dimension beträgt 32767.

Wird versucht, einen DIM-Befehl mehr als einmal für dieselbe Matrixvariable auszugeben, oder stößt GW-BASIC nach der impliziten Definition dieser Matrixvariablen (d.h. nach Benutzung der Matrixvariablen mit einem maximalen Index von 10 ohne vorhergehenden DIM-Befehl) auf DIM, so wird die Fehlermeldung "Duplicate Definition" (Doppelte Definition) angezeigt.

Beispiel: Siehe die praktischen Beispiele im Anschluß an den Abschnitt „Matrixvariablen" in Kapitel 1.

DRAW-Befehl (Grafik-Modus)

Syntax: DRAW <Zeichenfolgenausdruck>.

Verwendung: Zeichnet ein mit dem Zeichenfolgenausdruck angegebenes Objekt.

Bemerkungen: Mit dem DRAW-Befehl kann ein Objekt mit Befehlen in der Objektdefinitionssprache gezeichnet werden. Ein Sprachbefehl ist ein einzelnes Zeichen innerhalb einer Zeichenfolge, auf das wahlweise ein oder mehrere Parameter folgen. Der Zeichenfolgenausdruck definiert ein Objekt, das auf dem Bildschirm gezeichnet wird, wenn GW-BASIC den DRAW-Befehl ausführt.

Die folgenden Bewegungsbefehle beginnen die Bewegung ab den Koordinaten des letzten Punktes, auf den mit einem anderen Sprachbefehl oder einem anderen GW-BASIC Grafik-Befehl (z.B. LINE oder PSET) Bezug genommen wurde.

U [<n>]	Bewegung nach oben
D [<n>]	Bewegung nach unten
L [<n>]	Bewegung nach links
R [<n>]	Bewegung nach rechts
E [<n>]	Bewegung diagonal nach oben und rechts
F [<n>]	Bewegung diagonal nach unten und rechts
G [<n>]	Bewegung diagonal nach unten und links
H [<n>]	Bewegung diagonal nach oben und links

Das n in den vorhergehenden Befehlen gibt die Entfernung der Bewegung an. Die Anzahl von zurückgelegten Punkten entspricht n mal dem Skalenfaktor (siehe S unten). Wird n nicht angegeben, so erfolgt eine Bewegung um eine Skaleneinheit.

M<x,y> Absolute oder versetzte Bewegung (für eine Beschreibung der x- und y-Koordinaten wird auf Kapitel 3 verwiesen). Steht vor x ein + oder –, so

werden x und y zu den Koordinaten des letzten Punktes hinzugefügt, auf den Bezug genommen wurde. Der auf diese Weise angegebene Punkt wird mit dem letzten Punkt verbunden, auf den mit einer Linie Bezug genommen wurde. Wird kein + oder - hinzugefügt, so wird von dem letzten Referenzpunkt, eine Linie zu Punkt (x,y) gezogen.

Die folgenden Vorspann-Befehle können vor jedem der obigen Bewegungsbefehle stehen:

B
Bewegung, jedoch kein Zeichnen von Punkten.

N
Bewegung, jedoch Rückkehr zur Ausgangsposition, nachdem der Zeichenvorgang beendet ist.

Wird eine auf dem Bildschirm angezeigte Zeichnung betrachtet, so muß das „Seitenverhältnis" des Bildschirms berücksichtigt werden. Das Seitenverhältnis ist der vertikale Erweiterungsfaktor von 1.2 bei niedriger und hoher Grafikauflösung und von 2.4 bei mittlerer Grafikauflösung. In dem folgenden kurzen Programm zeichnet DRAW beispielsweise ein Quadrat als mittlerer Grafikauflösung:

```
10 SCREEN 2
20 DRAW "L96 D40 R96 U40"
```

Die senkrechten Seiten werden mit kleineren Werten als die waagerechten Seiten angegeben, da bei einer waagerechten Zeichnung mehr Bildelemente pro Zoll (2,4 mal soviel) dargestellt werden.

Bei einer Grafik mit niedriger Auflösung würde ein Quadrat folgendermaßen angegeben:

```
10 SCREEN 1
20 DRAW "L96 D80 R96 U80"
```

Neben dem Zeichnen einfacher gerader Linien, bietet DRAW noch die folgenden Grafikbefehle.

TA<n>
Dreht den Zeichenwinkel um <n> Grad entgegen dem Uhrzeigersinn. Wird für <n> ein

negativer Wert angegeben, so wird der Zeichenwinkel im Uhrzeigersinn gedreht. Aus

```
DRAW "TA5;U50"
```

führt dazu, daß sich die „obere" Linie um 5 Grad nach links neigt, wie auf dem Bildschirm ersichtlich. Liegt <n> außerhalb des Bereichs von –360 bis +360, so kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

A<n>

Entspricht dem TA-Befehl. Der einzige Unterschied liegt darin, daß <n> eine Zahl 0, 1, 2 oder 3 für 0, 90, 180 bzw. 270 Grad darstellt. Sowohl TA als auch A kompensieren eine Zeichnungserweiterung, zu der es ansonsten bei Angabe von 0 oder 180 Grad kommen würde. Siehe folgendes Beispiel.

C<n>

Bestimmt die Zeichenfarbe. Bei der Farbgrafik mit niedriger und hoher Auflösung kann <n> 1, 2 oder 3 darstellen, wobei dies die Farbe aus der gerade ausgewählten Palette ist. <n> kann jedoch auch 0 darstellen, die aktuelle Hintergrundfarbe (siehe COLOR). Wird keine Farbe angegeben, so benutzt GW-BASIC 3. Bei einfarbiger Grafik mittlerer und hoher Auflösung kann <n> 0 (schwarz) oder 1 (weiß) sein. Wird keine Farbe angegeben, so benutzt GW-BASIC 1.

S<n>

Legt den Skalenfaktor (Vergrößerungsfaktor) für die U, D, L, R, E, F, G, H und M Bewegungsbefehle (Versatzbefehle) fest. <n> dividiert durch 4 entspricht dem Skalenfaktor. Wird kein Skalenfaktor angegeben, so benutzt GW-BASIC 4 für <n>, d.h. einen Skalenfaktor 1. <n> muß ein ganzzahliger Wert in dem Bereich von 1 bis 255 sein. Der Skalenfaktor bleibt bis zum nächsten S<n> unter DRAW bzw. bis zum nächsten NEW in Kraft.

X<Zeichenfolgenvariable>
Stößt DRAW auf diesen Befehl, so führt er die in den „Variablen" stehenden Zeichenbefehle aus, bevor der Rest der Befehlsfolge verarbeitet wird. Auf diese Weise kann eine zweite Zeichenfolge innerhalb einer Zeichenfolge ausgeführt werden.

P<Umriß auszeichnen>
Zeichnet den Bereich der Grafik aus, in dem der letzte adressierte Punkt steht. Der Zeichnungsumriß muß schon farbig gezeichnet sein. Ansonsten geht die Zeichnung über die Umrißlinie hinaus. Bei Farbgrafiken mit niedriger und hoher Auflösung können die Farben 0, 1, 2 oder 3 sein. Bei einfarbiger Grafik mittlerer oder hoher Auflösung können sie 0 oder 1 sein (siehe „C<n>" oben). Sowohl für das „auszeichnen" als auch für den „Umriß" müssen Werte angegeben werden.

Zwischen den in einer DRAW-Befehlsfolge stehenden Elemente müssen keine Trennzeichen (nicht einmal Leerzeichen) stehen. Allerdings muß zwischen dem Namen einer Variablen und dem nächsten Element ein Semikolon stehen. Zwischen den Elementen können jedoch Leerzeichen und/oder Semikolon benutzt werden. Auf diese Weise ist die Zeichenfolge einfacher zu lesen.

Für sämtliche in dieser Beschreibung von DRAW angegebenen Werte von n und x,y sind Variablen zulässig. Vor einer Variablen steht ein Gleichzeichen, außer hinter dem X-Zeichenbefehl.

Beispiele:

```
10 SCREEN 1
20 DRAW „E15 F15 L30"
```

Zeichnet ein gleichschenkliges Dreieck.

```
10 SCREEN 1
20 V = 50
30 DRAW "U = V; R = V; D = V; L = V;"
```

Zeichnet ein Kästchen. (Diesmal wurden Variablen (V) benutzt, so daß ein Semikolon erforderlich ist.)

Die folgenden Zeilen bewegen den Punkt, auf den als letztes Bezug genommen wurde, in die Grenzen des Kästchens und zeichnen das Kästchen mit der Farbe 2 aus. Der Umriß wird mit Farbe 1 aus der gerade ausgewählten Palette gezeichnet:

```
30 DRAW "C2"
40 DRAW "BE10"
50 DRAW "P1,2"
```

Mit dem folgenden Beispiel werden Speichen gezeichnet:

```
10 SCREEN 1
20 FOR L%= 0 TO 360 STEP 10
30 DRAW "TA = L%;NU60"
40 NEXT L%
```

Zeichnen Sie nun einen Kreis mit demselben Mittelpunkt und dem Radius 60:

```
50 CIRCLE (160,100),60
```

so werden Sie feststellen, daß der Kreis für die Speichen zu klein ist. Um dies zu kompensieren, muß TA angewiesen werden, die erste Speiche in einer horizontalen Richtung zu zeichnen. Dies ist darauf zurückzuführen, daß der für den Kreis benutzte Radius eine Anzahl von horizontalen und nicht vertikalen Bildschirmpunkten darstellt (siehe CIRCLE).

Hinweis:

Mit der Funktion VARPTR$ können Variablen angegeben werden.

Wird mit DRAW über den Rand des Bildschirms hinaus gezeichnet, so kommt es zu keiner Fehlerbedingung (Ausnahme: TA<n>). Wird jedoch bei Grafiken mit niedriger oder hoher Auflösung über den rechten Rand des Bildschirms gezeichnet, so kommt es zu einem Umbruch zu der nächsten Horizontallinie.

Der X-Zeichenbefehl ist insofern nützlich, als Sie nach Bedarf Zeichnungen aufrufen können, die mehr als einmal benutzt werden sollen.

EDIT-Befehl

Syntax: EDIT<Zeilennummer>

Mit <Zeilennummer> wird die Nummer einer Programmzeile angegeben. Ist keine derartige Zeile vorhanden, so wird eine Fehlermeldung "Undefined Line Number" (Nichtdefinierte Zeilennummer) angezeigt.

Verwendung: Zeigt eine Zeile zur Editierung an.

Bemerkungen: Der EDIT-Befehl zeigt einfach die angegebene Zeile an und bewegt den Cursor unter die erste Ziffer der Zeilennummer. Danach kann die Zeile mit den in Kapitel 2, Gesamtbildschirmeditor, beschriebenen Tasten geändert werden.

Ein Punkt (.) bezieht sich stets auf die aktuelle Zeile. Wurde gerade eine Zeile eingegeben und soll zurückgegangen und diese Zeile editiert werden, so können Sie EDIT eingeben, um die Zeile erneut anzuzeigen.

Hinweis: Soll ein Block mit Programmzeilen zu Editierzwekken angezeigt werden, so können Sie den LIST-Befehl benutzen.

END-Befehl

Syntax: END

Verwendung: Dieser Befehl beendet die Programmausführung, schließt sämtliche Dateien ab und kehrt auf Befehlsebene ("Ok") zurück.

Bemerkungen: END kann an beliebiger Stelle in das Programm gesetzt werden, um die Ausführung zu beenden. Im Gegensatz zu dem STOP-Befehl führt END nicht zum Ausdrucken einer "Break"- (Unterbrechungs)-Meldung. Darüber hinaus schließt END sämtliche Dateien ab. Ein END-Befehl am Ende eines Programms ist wahlweise.

Beispiel:

```
520 IF K>1000 THEN END ELSE GOTO 20
```

ENVIRON-Befehl

Syntax: ENVIRON <Parameter-ID> [=Text]

Verwendung: Änderung der Parameter in der Umgebungszeichenfolgen-Tabelle von GW-BASIC, insbesondere des Parameters "PATH". Mit diesem Befehl kann Ihr Programm ein anderes Programm aufrufen (NCR-DOS nennt dies ein Mutter-Kind-Verfahren), selbst wenn das Programm in einem anderen Inhaltsverzeichnis steht (siehe SHELL).

Bemerkungen: Für Angaben über Pfad- und andere NCR-DOS Umgebungsparameter, wird auf das NCR-DOS Handbuch und das NCR-DOS Programmiererhandbuch verwiesen.

"Parameter-ID" ist der Name des Parameters, zum Beispiel PATH.

"Text" ist der neue Parametertext. Wird "Text" weggelassen, so wird der Parameter aus der Umgebungszeichenfolgen-Tabelle gelöscht und die Tabelle komprimiert. Wird "Text" angegeben, so muß er in Anführungszeichen stehen. Vor ihm muß ein Gleichzeichen oder ein Leerzeichen stehen. Enthält "Text" nur ein Semikolon, so wird er als nicht vorhanden betrachtet.

Beispiel: Mit dem folgenden Befehl kann GW-BASIC auf das Inhaltsverzeichnis SALES in Laufwerk A zugreifen.

```
ENVIRON "PATH = A:\ SALES"
```

Hinweis: ENVIRON läßt nur Zeichenfolgen als Parameter zu. Werden keine Zeichenfolgen benutzt, so wird eine Fehlermeldung "Type mismatch" (Übereinstimmungsfehler) angezeigt. "Out of memory" (Kein Speicherplatz) gibt an, daß die Umgebungszeichenfolgen-Tabelle voll ist.

Die Umgebungszeichenfolgen-Tabelle von GW-BASIC ist ursprünglich leer, es sei denn, sie wird mit dem PATH-Befehl von NCR-DOS geändert.

ENVIRON$-Funktion

Syntax:

ENVIRON$ (<"Parameter-ID">)
ENVIRON$ (n)

Verwendung:

Rückübertragung eines Umgebungs-Parameters aus der Umgebungszeichenfolgen-Tabelle von GW-BASIC.

Bemerkungen:

"Parameter-ID" ist der Name des Parameters in Anführungszeichen.

n ist ein ganzzahliger Ausdruck, der einen Wert in dem Bereich von 1 bis 255 aufweist. Dieser Wert stellt den n-ten Parameter in der Tabelle dar.

Wird der Parameter nicht gefunden, so gibt die ENVIRON$-Funktion eine leere Zeichenfolge zurück.

Beispiele:

Hier soll davon ausgegangen werden, daß der einzige Parameter in der Tabelle der Parameter ist, der in dem Beispiel für den ENVIRON-Befehl zugewiesen wurde.

PRINT ENVIRON$ („PATH")

zeigt

A:\ SALES

an. Der Befehl

PRINT ENVIRON$(1)

zeigt

PATH = A:\ SALES

an. Mit dem folgenden Programm wird die Umgebungszeichenfolgen-Tabelle von GW-BASIC in einer Matrix gesichert, damit sie für ein Mutter-Kind-Verfahren geändert werden kann. Nachdem das Mutter-Kind-Verfahren beendet ist, wird die Umgebungstabelle wieder hergestellt.

```
10 DIM ENV.TBL$(10) 'Nicht mehr als
   10 Parameter
```

```
 20 N.PARMS = 1 'Ursprüngliche Anzahl
    von Parametern
 30 WHILE LEN(ENVIRON$(N.PARMS)) > 0
 40 ENV.TBL$(N.PARMS)=
    ENVIRON$(N.PARMS)
 50 N.PARMS= N.PARMS+1
 60 WEND
 70 N.PAMS = N.PARMS-1 'an richtige Zahl
    anpassen
 80 'Jetzt neue Umgebungs-Tabelle speichern
 90 ENVIRON "MYCHILD.PARM1 = SORT
    BY NAME"
100 ENVIRON "MYCHILD.PAR2 = LIST BY
    NAME"
                .
                .
                .
1000 SHELL "MYCHILD" 'Führt
    "MYCHILD.EXE" aus
1010 FOR I = 1 TO N.PARMS
1020 ENVIRON ENV.TBL$(I) 'Parameter
    wiederherstellen
1030 NEXT I
                .
                .
                .
```

Hinweis:

Ist "Parameter-Id" keine Zeichenfolge, so kommt es zu einer Fehlermeldung "Type mismatch" (Übereinstimmungsfehler). Ist die Zeichenfolge zu lang, so wird die Fehlermeldung "String too long" (Zeichenfolge zu lang) angezeigt. Sind zu wenig Parameter in der Tabelle für einen sinnvollen Einsatz von "n" vorhanden, so kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

EOF-Funktion

Syntax: EOF (<Dateinummer>)

Verwendung: Testet, ob das Ende einer Datei erreicht ist.

Bemerkungen: EOF gibt einen wahren Wert (-1) zurück, wenn keine weiteren Daten mehr in der Datei stehen. Die Datei ist leer, wenn die nächste Eingabeoperation (z.B. INPUT#, LINE INPUT#) zu einer Fehlermeldung "Input past end" (Eingabe über Ende hinaus) führen würde.

Bei einer Übertragungsdatei wird der Wert zurückgegeben, wenn der Eingabepuffer leer ist.

Die EOF-Bedingung ist für Direktzugriffsdateien nicht signifikant.

Beispiel: Bei diesem Beispiel wird jeder Satz einer sequentiellen Datei "NAMES" angezeigt, die schon vorhanden ist. Das Dateiende wird entdeckt, sobald es erreicht ist, so daß eine Fehlersituation vermieden wird.

```
10 OPEN "NAMES" FOR INPUT AS #1
20 IF EOF(1) THEN PRINT "Das ist alles": END
30 INPUT #1, N$
40 PRINT N$ 50 GOTO 20
```

Hinweis: EOF kann auch für eine EIA benutzt werden, die auf Standard-Eingabeeinheiten umgeleitet wird. In diesem Fall wird 0 als "Dateinummer" angegeben.

ERASE-Befehl

Syntax: ERASE <Liste mit Matrixvariablen>

Verwendung: Löscht Matrizen aus dem Speicher.

Bemerkungen: Matrizen können neu dimensioniert werden, nachdem sie gelöscht wurden. Der vorher den Matrizen im Speicher zugewiesene Platz kann jedoch auch für andere Zwecke benutzt werden. Wird versucht, eine Matrix neu zu dimensionieren, ohne sie zuerst zu löschen, so wird eine Fehlermeldung "Duplicate definition" (Doppelte Definition) angezeigt.

Beispiel: In dem folgenden Programm wird mit der FRE-Funktion gezeigt, wieviel Platz gespart werden kann, wenn eine große Matrixvariable, die nicht mehr benötigt wird, mit ERASE gelöscht wird.

```
10 PRINT "Freie Bytes vor dem Dimensionieren
   einer großen Matrix mit DIM:";FRE("„)
20 DIM DINOSAUR (100,100)
30 PRINT "Freie Bytes nach Dimensionieren
   einer großen Matrix mit DIM:";FRE("")
40 ERASE DINOSAUR
50 DIM DINOSAUR (10,10)
60 PRINT "Dies ist nun eine wesentlich kleinere
   Matrix. Freie Bytes:";FRE("")
```

Sie sehen, daß die neu dimensionierte Matrix ca. 40000 Bytes weniger Speicherplatz benötigt, als die große Matrix.

Hinweis: Mit CLEAR werden sämtliche Programmvariablen gelöscht.

ERR- und ERL-Systemvariablen

Verwendung: Mit diesen Systemvariablen wird festgestellt, an welcher Stelle in dem Programm es zu einem Fehler gekommen ist. Außerdem wird die Art dieses Fehlers ermittelt.

Bemerkungen: Wird eine Fehlerbearbeitungsroutine eingegeben, so enthält die Variable ERR den Fehlercode für den Fehler. Die Variable ERL enthält die Zeilennummer der Zeile, in der der Fehler entdeckt wurde. Die Variablen ERR und ERL werden normalerweise in IF...THEN Entscheidungen benutzt, um den Programmfluß in der Fehlerbearbeitungsroutine zu steuern. Die GW-BASIC Fehlercodes werden in Anhang C aufgelistet.

Wurde der den Fehler erzeugende Befehl im direkten Modus eingegeben, so enthält ERL 65535. Soll getestet werden, ob ein im direkten Modus eingegebener Befehl den Fehler verursacht hat, so geben Sie

```
IF 65535 = ERL THEN.....
```

ein. Ansonsten wird ERL links neben den Vergleichsoperator (z.B. =) geschrieben, damit die auf der rechten Seite angegebene Seitennummer während der Programmeditierung nicht von dem RENUM-Befehl ausgelassen wird, z.B.:

```
IF ERL<20 THEN PRINT „Der Fehler ist in einer Zeile ziemlich am Anfang des Programms aufgetreten“
```

Beispiel: Siehe ON ERROR

Hinweis: ERR und ERL sind System- und keine Programm-Variablen. Sie können Ihnen keine Werte zuweisen. Sie können nur die von GW-BASIC in diese Variablen gesetzten Werte betrachten.

ERROR-Befehl

Syntax: ERROR <ganzzahliger Ausdruck>

Verwendung: Simuliert das Auftreten eines GW-BASIC Fehlers oder ermöglicht die Definition von Fehlercodes.

Bemerkungen: Der Wert von <ganzzahliger Ausdruck> muß größer sein als 0 und kleiner als 255. Entspricht der Wert von <ganzzahliger Ausdruck> einem schon von GW-BASIC benutzten Fehlercode (siehe Anhang C), so simuliert der ERROR-Befehl das Auftreten dieses Fehlers. Die entsprechende Fehlermeldung wird ausgedruckt. (Siehe Beispiel 1.)

Zur Definition Ihres eigenen Fehlercodes benutzen Sie einen Wert, der größer ist als einer der von GW-BASIC benutzten Fehlercodes. (Es wird empfohlen, die höchsten verfügbaren Werte zu benutzen, damit die Kompatibilität aufrecht erhalten werden kann, wenn weitere Fehlercodes zu GW-BASIC hinzugefügt werden.) Dieser vom Benutzer definierte Fehlercode kann dann bequem in einer Fehlerbearbeitungsroutine verarbeitet werden. (Siehe Beispiel 2.)

Wird mit dem ERROR-Befehl ein Code angegeben, für den keine Fehlerbearbeitung definiert wurde, so antwortet GW-Basic mit der Fehlermeldung "Unprintable error" (Nicht ausdruckbarer Fehler) und beendet die Programmausführung.

Beispiel 1

```
10 S = 10
20 T = 5
30 ERROR S+T
40 END
```

ergibt

```
Zeichenfolge zu lang in Zeile 30
```

Beispiel 2

```
.
.
.
110 ON ERROR GOTO 400
120 INPUT "WIE LAUTET IHRE WETTE";B
130 IF B>5000 THEN ERROR 210
```

```
.
.
.
400 IF ERR=210 THEN PRINT "INTERNES
    LIMIT BETRÄGT $5000".
410 IF ERL = 130 THEN RESUME 120
.
.
.
```

Hinweis:

ERROR ist beim Testen der Fehlerbearbeitungsroutinen besonders nützlich. Wie die meisten anderen Befehle kann ERROR im direkten Modus eingegeben werden.

EXP-Funktion

Syntax: EXP(X)

Verwendung: Erhebt e (Basis natürlicher Logarithmen) in die X-te Potenz. X muß $<=$ 88.02969 sein.

Bemerkungen: Ist x größer als 88.02969, so wird die Fehlermeldung "Overflow" (Überlauf) angezeigt, die Ausführung wird jedoch fortgesetzt.

Beispiel:

```
10 X = 5
20 PRINT EXP(X-1)
ergibt
 54.59815
```

FIELD-Befehl

Syntax: FIELD [#] <Dateinummer>,<Feldbreite> AS <Zeichenfolgenvariable>...

Verwendung: Weist Platz für Variablen in einem Puffer für Direktzugriffsdateien zu.

Bemerkungen: Bevor GET oder PUT ausgeführt werden können, muß ein FIELD-Befehl ausgeführt werden, um den Puffer für die Direktzugriffsdateien zu formatieren.

<Dateinummer> ist die Nummer, unter der die Datei eröffnet wurde. <Feldbreite> ist die Anzahl von Zeichen, die <Zeichenfolgenvariable> zugewiesen werden müssen.

Die Gesamtanzahl von Bytes, die in einem FIELD-Befehl zugewiesen werden, darf die beim Eröffnen der Datei angegebene Satzlänge nicht überschreiten. Ansonsten wird eine Fehlermeldung "Field overflow" (Feldüberlauf) angezeigt. (Die Standardsatzlänge beträgt 128 Bytes.)

Eine beliebige Anzahl von FIELD-Befehlen kann für dieselbe Datei ausgeführt werden. Sämtliche FIELD-Befehle, die ausgeführt wurden, bleiben in dem gesamten Programm wirksam.

FIELD setzt nicht wirklich Daten in den Puffer für die Direktzugriffsdateien (dies erfolgt über LSET und RSET) und ruft auch keine Daten aus der Direktzugriffsdatei auf (dies erfolgt über GET).

Beispiele: FIELD 1,20 AS N$, 10 AS ID#, 40 AS ADD$, 58 AS LEFTOVER$

Weist GW-BASIC an, daß nach Lesen eines Datensatzes aus der Direktzugriffsdatei mit der Nummer 1 die 20 ersten Bytes dieses Datensatzes als zu der Zeichenfolgenvariable N$ gehörig, die 10 nächsten Bytes als zu ID$ gehörig usw. betrachtet werden können. Zur Anzeige der 20 ersten Bytes könnte der Befehl PRINT N$ ausgegeben werden.

Diesen Variablen darf kein Inhalt (beispielsweise mit LET oder INPUT) zugewiesen werden, da die-

ser ihnen schon in dem Puffer für die Direktzugriffsdateien zugewiesen wurde.

Soll ein Satz in eine Direktzugriffsdatei geschrieben werden, so werden die Daten mit dem LSET- (oder RSET)-Befehl in den Direktzugriffspuffer gesetzt. Mit dem PUT-Befehl wird der Inhalt des Puffers auf Diskette geschrieben. In dem folgenden Beispiel werden ein neuer Telefonteilnehmer und seine Telefonnummer als vierter Satz in die Datei geschrieben. (Mit LSET werden die Daten linksbündig ausgerichtet in den Bereich des Puffers gesetzt, der durch die Feldvariable begrenzt wird.)

```
200 OPEN "R", #1, "TELNUMS",35
210 FIELD 1,25 AS NNAME$,10 AS
    PHONENO$
220 LSET NNAME$="ISAAC NEWTON"
230 LSET PHONENO$="1234"
240 PUT 1,4
250 END
```

FILES-Befehl

Syntax: FILES [<Dateispez>]

Verwendung: Ausdrucken der Namen von Dateien, die auf der angegebenen Diskette stehen.

Bemerkungen: Wird <Dateispez> weggelassen, so werden sämtliche Dateien in dem gegenwärtigen Inhaltsverzeichnis in dem gerade ausgewählten Laufwerk aufgelistet. <Dateispez> ist eine Zeichenfolge, die Fragezeichen (?) oder Sternchen (*) als Dateigruppensymbole enthalten kann. Ein Fragezeichen entspricht einem beliebigen einzelnen Zeichen in dem Dateinamen oder der Dateinamenerweiterung. Ein Sternchen entspricht einem oder mehreren Zeichen ab dieser Zeichenposition. Das Sternchen ist eine Abkürzung für eine Folge von Fragezeichen. Es wird empfohlen, auch einen Pfadnamen in "Dateispez" aufzunehmen.

Beispiele: FILES

zeigt sämtliche Dateien in dem gegenwärtigen Inhaltsverzeichnis auf dem gerade ausgewählten Laufwerk an.

FILES "*.BAS"

zeigt sämtliche Dateien mit der Erweiterung .BAS in dem gegenwärtigen Inhaltsverzeichnis des gerade ausgewählten Laufwerks an.

FILES "B:*.*" oder FILES "B:"

zeigt sämtliche Dateien in Laufwerk B an.

FILES "TEST?.BAS"

zeigt sämtliche aus fünf Buchstaben bestehenden Dateien, deren Namen mit "TEST" beginnen und mit der .BAS-Erweiterung enden, in dem gegenwärtigen Inhaltsverzeichnis des gerade ausgewählten Laufwerks an.

Die Dateinamen werden in einem Format angezeigt, das die Position einer Datei in ihrer unmittel-

bare Umgebung in den hierarchischen Inhaltsverzeichnissen von NCR-DOS angibt. Handelt es sich bei dem Dateinamen in Wirklichkeit um ein Unterverzeichnis, so wird dies durch "<DIR>" im Anschluß an den Verzeichnisnamen angegeben. Wird auf die in der folgenden Abbildung dargestellte hierarchische Struktur Bezug genommen, so zeigt der Befehl

FILES " \ VERKAUF"

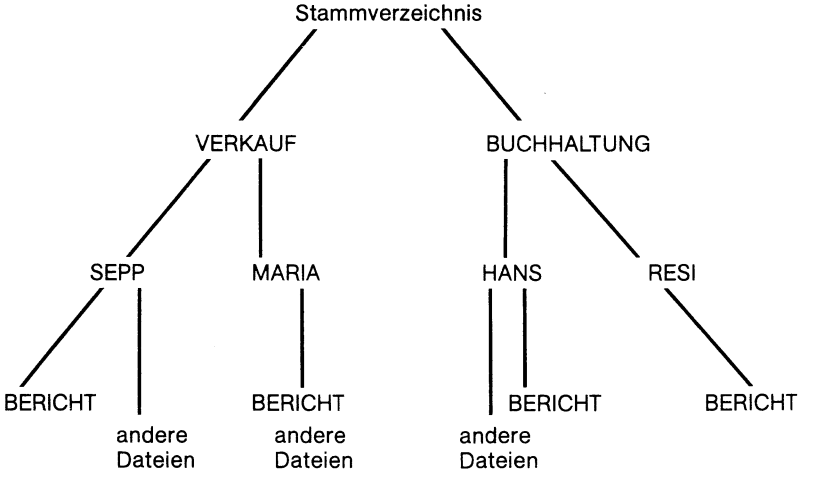
den Verzeichniseintrag

VERKAUF <DIR>

an. Der Befehl

FILES " \ VERKAUF \ MARIA \ "

zeigt die Namen sämtlicher Dateien in dem Inhaltsverzeichnis namens MARIA an.



FIX-Funktion

Syntax: FIX(X)

Verwendung: Gibt den Wert der Ziffern links von dem Dezimalpunkt in der Zahl X an, und ignoriert eventuelle Ziffern rechts von dem Dezimalpunkt.

Bemerkungen: Der Unterschied zwischen FIX und INT liegt darin, daß FIX die nächstniedrige Zahl bei negativen X nicht zurückgibt (siehe INT,CINT).

Beispiele:

```
PRINT FIX(58.75)
ergibt
 58

PRINT FIX(-58.75)
ergibt
 -58
```

FOR...NEXT-Befehl

Syntax:

FOR <Variable> = x TO y [STEPz]

.
.
.

NEXT [<Variable>] [,<Variable>...]

wobei x, y und z numerische Ausdrücke darstellen.

Verwendung:

Mit diesem Befehl kann eine Reihe von Instruktionen eine bestimmte Anzahl von Malen in einer Schleife ausgeführt werden.

Bemerkungen:

"Variable" wird als Zähler benutzt. Der erste numerische Ausdruck (x) ist der Ausgangswert des Zählers. Der zweite numerische Ausdruck (y) ist der Endwert des Zählers. Die auf den FOR-Befehl folgenden Programmzeilen werden ausgeführt, bis NEXT angetroffen wird. Danach wird der Zähler an den von STEP angegebenen Wert angepaßt. Anschließend wird geprüft, ob der Wert des Zählers nun größer ist als der endgültige Wert (y). Ist er nicht größer, so geht GW-BASIC zu dem auf den FOR-Befehl folgenden Befehl zurück und das Verfahren wird wiederholt. Ist der Wert größer als der Endwert, so wird die Ausführung mit dem auf NEXT folgenden Befehl fortgesetzt. Diese Reihe von Ereignissen wird häufig als eine FOR...NEXT-Schleife bezeichnet.

Wird STEP nicht angegeben, so wird von einer Erhöhung von 1 ausgegangen. Ist STEP negativ, so wird der Zähler nach jeder Ausführung der Schleife durch GW-BASIC vermindert. Die Schleife wird solange ausgeführt, bis der Zähler einen niedrigeren Wert als den endgültigen Wert enthält.

Bei dem Zähler muß es sich um eine ganzzahlige oder numerische Konstante mit einfacher Genauigkeit handeln. Wird eine numerische Konstante mit doppelter Genauigkeit benutzt, so kommt es zu einer Fehlermeldung "Type mismatch" (Übereinstimmungsfehler). Die Benutzung einer Ganzzahl als Zähler führt zu einer besseren Programmleistung.

Der Inhalt der Schleife wird übersprungen, wenn der Ausgangswert der Schleife größer ist als der endgültige Wert, wobei von einem positiven Wert für STEP ausgegangen wird. Enthält STEP einen negativen Wert, so wird der Inhalt der Schleife übersprungen, wenn der Ausgangswert kleiner ist als der endgültige Wert.

Verschachtelte Schleifen

FOR...NEXT-Schleifen können verschachtelt sein, d.h., eine FOR...NEXT-Schleife kann innerhalb einer anderen FOR...NEXT-Schleife stehen. Bei verschachtelten Schleifen muß jede Schleife einen einmaligen Variablennamen als Zähler aufweisen. Der NEXT-Befehl für die innere Schleife muß vor dem NEXT-Befehl für die äußere Schleife stehen. Verfügen verschachtelte Schleifen über denselben Endpunkt, so kann ein einziger NEXT-Befehl für sämtliche Schleifen benutzt werden.

Die Variablen für NEXT können weggelassen werden. In diesem Fall stimmt der NEXT-Befehl mit dem letzten FOR-Befehl überein. Wird ein NEXT-Befehl vor dem entsprechenden FOR-Befehl angetroffen, so wird die Fehlermeldung "NEXT without FOR" (NEXT ohne FOR) ausgegeben und die Ausführung beendet.

Wird der FOR-Variablenname mit dem entsprechenden NEXT benutzt, so kommt es zu einer geringfügigen Verringerung der Ausführungsgeschwindigkeit, das Programm ist jedoch wesentlich einfacher zu lesen.

Beispiel 1

```
10 K = 10
20 FOR I = 1 TO K STEP 2
30 PRINT I;
40 K = K+10
50 PRINT K
60 NEXT
ergibt
 1   20
 3   30
 5   40
```

```
 7   50
 9   60
```

Beispiel 2

```
10 J = 0
20 FOR I = 1 TO J
30 PRINT I
40 NEXT I
50....
```

In diesem Beispiel wird die Schleife nicht ausgeführt, da der Ausgangswert der Schleife den Endwert überschreitet. Das Programm springt zu Zeile 50.

Beispiel 3

```
10 I = 5
20 FOR I = 1 TO I+5
30 PRINT I;
40 NEXT
```

ergibt

```
1 2 3 4 5 6 7 8 9 10
```

In diesem Beispiel wird die Schleife zehn Mal ausgeführt. Der Endwert für die Schleifenvariable wird stets vor dem Ausgangswert gesetzt.

Beispiel 4

Die Vielseitigkeit einer FOR...NEXT-Schleife wird am besten deutlich, wenn sie für die Adressierung der Elemente einer Matrixvariablen benutzt wird. Mit dem folgenden Programm können die Namen von sechs Tieren eingegeben werden. Für jedes Tier kann aus drei Menüs gewählt werden. In dem ersten Teil können die Tiere und ihr Lieblingsfutter eingegeben werden. Danach werden diese Informationen in ZOO$ gespeichert.

```
10 OPTION BASE 1
20 DIM ZOO$(6,4)
30 CLS
40 INPUT "Heutiges Datum"; DAY$
50 CLS
60 LOCATE 23,33:PRINT "Das Tier-Menü"
70 FOR A%= 1 TO 6
80 PRINT
100 INPUT "Welches Tier";ZOO$(A%,1)
110 FOR M%= 2 TO 4
```

```
130 PRINT "Menü";M%-1;" für
    ";ZOO$(A%,1)"? ";
140 INPUT ZOO$(A$,M%)
150 NEXT M%
160 NEXT A%
```

Im zweiten Teil wird das gesamte Menü für ZOO angezeigt.

```
170 CLS
180 LOCATE 1,26
190 PRINT "Das Tier-Menü von heute"
    ,,,,DAY$
210 FOR A%= 1 TO 6
220 PRINT
230 PRINT "Das";ZOO$(A%,1);"kann auswählen
    aus",
240 FOR M%= 2 TO 4
250 PRINT ZOO$(A%,M%),
260 NEXT M%
```

FRE-Funktion

Syntax:

FRE(0)
FRE("")

Verwendung:

Ermittelt den noch im Speicher freien Platz und spart Platz bei den Zeichenfolgen.

Bemerkungen:

Mit einem numerischen Parameter gibt FRE die Anzahl von Bytes im Speicher zurück, die nicht von GW-BASIC benutzt werden. Die Parameter bei FRE sind Pseudoparameter; d.h., die Syntax setzt das Vorhandensein dieser Parameter voraus, sie werden jedoch nicht von der Funktion verarbeitet.

FRE("") gibt Speicherplatz frei, der von Zeichenfolgen belegt wird, die nicht mehr benötigt werden, bevor die Anzahl von freien Bytes zurückgegeben wird.

GW-BASIC leitet erst dann eine Freigabe des Speicherplatzes ein, wenn nahezu der ganze freie Speicherplatz benutzt wurde. Tritt dies erst spät ein, so kann es einige Zeit dauern, bis Speicherplatz freigegeben wird. Deshalb wird empfohlen, FRE in regelmäßigen Abständen zu benutzen, um die Fristen zu verkürzen.

Beispiel:

Siehe ERASE

Hinweis:

Bei der Anzahl von Bytes, die von der FRE-Funktion zurückgegeben werden, wird der von dem GW-BASIC Interpreter im Speicher belegte Arbeitsplatz nicht berücksichtigt. Selbst wenn keine Daten im Arbeitsbereich stehen, reserviert GW-BASIC zwischen 2,5 KB und 4 KB.

GET-(Dateien)-Befehl

Syntax: GET [#] <Dateinummer> [,<Satznummer>]

Verwendung: Liest einen Satz aus einer Direktzugriffsdatei in einen Puffer für Direktzugriffsdateien.

Bemerkungen: <Dateinummer> ist die Nummer, unter der die Datei mit OPEN eröffnet wurde. Wird "Satznummer" weggelassen, so wird der nächste Satz (hinter dem letzten GET) in den Puffer gelesen. Die größtmögliche Satznummer lautet 16,777,215; die kleinste Nummer ist 1. "Satznummer" kann die Form eines mathematischen Ausdrucks oder eines Variablennamens aufweisen.

Nachdem der Satz in dem Puffer steht, kann das Programm ihn mit INPUT#, LINE INPUT# oder durch Bezugnahme auf die Variablen lesen, die in einer FIELD-Definition für den Puffer benutzt werden.

Beispiel: Siehe FIELD

Hinweis: Mit GET können auch Bytes aus einer Datenübertragungsdatei gelesen werden. In diesem Fall bezieht sich "Satznummer" nicht auf die Sätze; statt dessen stellt es die Anzahl von Bytes dar, die aus dem Datenübertragungspuffer gelesen werden müssen, vorausgesetzt, diese Zahl ist nicht größer als der Wert, der von der Option LEN bei OPEN "COM... festgelegt wurde.

GET-(Grafik)-Befehl

Syntax: GET (x1,y1)-(x2,y2),Matrix

Verwendung: Liest Grafikinformationen vom Bildschirm (nur Grafik-Modus) in eine Matrixvariable.

Bemerkungen: x1,y1 und x2,y2 sind die entgegengesetzten Ecken eines imaginären Rechtecks. Die Farbe jedes Punktes innerhalb dieses Rechtecks wird in die angegebene Matrix gelesen.

Mit der Gleichung

BYTES = 4 + INT
((XLEN*RESOLUTION+7)/8)*YLEN

wird die erforderliche Größe der Matrix in Bytes angegeben. XLEN stellt die horizontale Länge des Rechteckes und YLEN die vertikale Länge dar. RESOLUTION beträgt bei der Farbgrafik mit niedriger und hoher Auflösung 2 und bei der einfarbigen Grafik mit mittlerer und hoher Auflösung 1. (Hier handelt es sich um die Anzahl von Bits, die für die Speicherung eines Bildschirmpunktes in den Video-RAM erforderlich sind.)

Soll beispielsweise eine Grafik mit niedriger Auflösung mit einer Größe von 15 Bildelementen in horizontaler und 12 Bildelementen in vertikaler Richtung gespeichert werden, so ist die Anzahl von erforderlichen Bytes:

4+INT((15*2+7)/8)*12

Dies führt zu einem Ergebnis von 60 Bytes.

Nun brauchen Sie nur noch zu entscheiden, in welcher Art von numerischer Matrix die Grafik gespeichert werden soll. In dem Abschnitt über den erforderlichen Platzbedarf in Kapitel 1 wurden die Bytes pro Element einer Matrix folgendermaßen angegeben:

- Ganzzahlige Matrix – 2
- Matrix mit einfacher Genauigkeit – 4
- Matrix mit doppelter Genauigkeit – 8

Dies bedeutet, daß eine ganzzahlige Matrix mit 26 Elementen zur Speicherung der 15x12 Grafik groß genug ist. Wird eine ganzzahlige Matrix anstelle einer Matrix mit einfacher oder doppelter Genauigkeit benutzt, so bietet dies den Vorteil, daß die horizontale und vertikale Dimension der Grafik überprüft werden kann. Das erste Element enthält die horizontale Länge und das zweite die vertikale Länge.

Beispiel:

Mit dem folgenden Programm wird ein 15x12 Rechteck mit niedriger Auflösung von der oberen linken Ecke des Bildschirms in die Matrixvariable A% ge-speichert. Der Inhalt der beiden ersten Elemente der Matrix wird angezeigt.

```
10 DIM A%(52)
20 SCREEN 1
30 GET (0,0)-(14,11)
40 SCREEN 0:WIDTH 80
50 PRINT A%(0),A%(1)
```

Die linksbündigste der beiden angezeigten Zahlen entspricht der Horizontallänge mal 2; die rechtsbündige Zahl entspricht der vertikalen Länge des Rechteckes.

```
30   12
```

Ändern Sie nun Zeile 20 in SCREEN 2 (vergessen Sie nicht, <CR> zu betätigen). Mit dieser Instruktion wird GW-BASIC angewiesen, Grafiken mit mittlerer Auflösung zu benutzen. Nun wird das Programm mit RUN erneut gestartet. Diesmal stellen die beiden angezeigten Zahlen die horizontale und vertikale Länge des Rechteckes dar:

```
15   12
```

Hinweis:

Mit dem Zusatzbefehl PUT kann der Inhalt einer Matrix auf dem Bildschirm angezeigt werden. Sowohl GET als auch PUT können effizienter eingesetzt werden, wenn es sich bei x1 um eine Zahl handelt, die ohne Rest durch 8 (bei Farbgrafik mit niedriger und hoher Auflösung) oder durch 16 (bei einfarbiger Grafik mittlerer und hoher Auflösung) geteilt werden kann.

Außerdem können verschobene Koordinaten benutzt werden. Mit

```
GET (100,100)-STEP(15,-12),A%
```

wird beispielsweise festgelegt, daß die Grafikinformationen eines Rechtecks, dessen obere linke Ecke bei dem Punkt 100,100 liegt, in die Matrixvariable A% gelesen werden müssen.

GOSUB...RETURN-Befehl

Syntax:

GOSUB<Zeilennummer>

.
.
.

RETURN [<Zeilennummer>]

Verwendung:

Verzweigung zu und Rücksprung aus einer Subroutine.

Bemerkungen:

<Zeilennummer> in dem GOSUB-Befehl entspricht der ersten Zeile der Subroutine.

Eine Subroutine kann beliebig oft in einem Programm aufgerufen werden. Eine Subroutine kann auch aus einer anderen Subroutine heraus aufgerufen werden. Eine derartige Verschachtelung von Subroutinen wird nur durch den zur Verfügung stehenden Speicherplatz beschränkt.

Durch einfache RETURN-Befehle in einer Subroutine geht GW-BASIC wieder zu dem Befehl zurück, der auf den als letztes angetroffenen GOSUB-Befehl folgt. Eine Subroutine kann mehr als einen RETURN-Befehl enthalten. Deshalb braucht sich also GW-BASIC nicht zu der letzten Zeile der Subroutine zu verzweigen, um zurückzuspringen.

Mit der Option <Zeilennummer>in dem RETURN-Befehl kann aus der Subroutine zu einer bestimmten Zeilennummer zurückgegangen werden. Bei dieser Art von Rücksprung muß jedoch vorsichtig vorgegangen werden, da zum Zeitpunkt des GOSUB-Befehls aktive GOSUB-, WHILE- oder FOR-Befehle aktiv bleiben, und es zu Fehlermeldungen wie beispielsweise "FOR without NEXT" (FOR ohne NEXT) kommen kann.

Subroutinen können an beliebiger Stelle in dem Programm stehen. Es wird jedoch empfohlen, daß die Subroutine eindeutig von dem Hauptprogramm unterschieden wird. So wird beispielsweise eine Subroutine häufig mit einer REM-Zeile begonnen, in der die Funktion der Subroutine angegeben wird. Um einen versehentlichen Eintritt in die Subrou-

tine zu vermeiden, sollte vor ihr ein STOP-, END- oder GOTO-Befehl stehen, der die Programmsteuerung um die Subroutine herumleitet.

Beispiel:

```
10 GOSUB 40
20 PRINT "RÜCKSPRUNG AUS
   SUBROUTINE"
30 END
40 PRINT "SUBROUTINE";
50 PRINT "IN";
60 PRINT "VERARBEITUNG"
70 RETURN
```

ergibt

```
SUBROUTINE IN VERARBEITUNG
RÜCKSPRUNG AUS SUBROUTINE
```

Hinweis:

Mit dem ON...GOSUB-Befehl kann eine Subroutine je nach Ergebnis einer vorhergehenden Operation ausgewählt werden.

GOTO-Befehl

Syntax: GOTO<Zeilennummer>

Zweck: Unbedingte Verzweigung aus der normalen Programmfolge zu einer angegebenen Zeilennummer.

Bemerkungen: Handelt es sich bei <Zeilennummer> um die Zeilennummer eines ausführbaren Befehls, so werden dieser Befehl und die darauf folgenden Befehle ausgeführt. Handelt es sich um einen nicht-ausführbaren Befehl (z.B. REM), so fährt die Programmausführung mit dem ersten ausführbaren Befehl hinter <Zeilennummer> fort.

Beispiel:

```
10 READ R
20 PRINT "R = ";R,
30 A=3.14*R^2
40 PRINT "AREA =";A
50 GOTO 10
60 DATA 5,7,12
ergibt
R = 5        AREA = 78.5
R = 7        AREA = 153.86
R = 12       AREA = 452.16
Out of data in 10
```

Mit dem Befehl GOTO 10 wird eine unendliche Schleife festgelegt. GW-BASIC geht erst dann aus dieser Schleife, wenn keine weiteren DATEN mehr zu LESEN sind. (In Wirklichkeit wird die Schleife nur drei Mal ausgeführt.)

Das folgende Beispiel stellt eine echte unendliche Schleife dar.

```
10 THRU% = 1
20 PRINT "Dies ist eine Ausführungsnummer";
   THRU"; "durch die Schleife"
30 THRU% = THRU% +1
40 GOTO 20
```

Der Computer muß jedoch nicht ausgeschaltet werden, um aus der Schleife zu gehen. In Situationen wie dieser sieht GW-BASIC eine Unterbrechungsmöglichkeit vor, nämlich die Tastenkombination

<Ctrl-Break>. Bei dieser Unterbrechung bleiben sogar die Variablen intakt, so daß sie (mit PRINT-....im Direkt-Modus) überprüft werden können.

Hinweis:

Durch Eingabe von GOTO mit einer Zeilennummer im Direkt-Modus kann nach einer Unterbrechung in der Ausführung am Anfang einer beliebigen Zeile wieder in ein Programm gegangen werden. Hierbei spielt es keine Rolle, ob Sie das Programm (mit STOP-Befehl oder <Ctrl-Break>) unterbrochen haben oder ob GW-BASIC aufgrund eines Fehlers stoppen mußte.

Mit dem ON...GOTO-Befehl kann je nach Ergebnis einer vorhergehenden Operation eine Programmzeile ausgewählt werden, zu der sich GW-BASIC verzweigen muß.

HEX$-Funktion

Syntax: HEX$(X)

Verwendung: Rückgabe einer Zeichenfolge, die den Hexadezimalwert des Dezimalparameters darstellt.

Bemerkungen: Ist X noch keine Ganzzahl, so wird sie vor Auswertung von HEX$(X) in eine Ganzzahl aufgerundet. Diese Ganzzahl muß in dem Bereich von -32768 bis 65535 liegen.

Beispiel:

```
10 INPUT X
20 A$=HEX$(X)
30 PRINT X "DEZIMALZAHL IST" A$"
   HEXADEZIMALWERT"
ergibt
? 32
32 DEZIMALZAHL IST 20
   HEXADEZIMALWERT
```

Hinweis: Ist der Wert von X negativ, so gibt HEX$ das "Zweier-Komplement" zurück. Dies ist dasselbe, als würde X als positiv angesehen und die HEX$-Funktion auf die Differenz zwischen 65536 und X angewandt.

Für die Umwandlung von Hexadezimalzahlen in Dezimalzahlen wird auf Anhang E verwiesen.

IF...THEN...ELSE...
IF...GOTO Befehle

Syntax:

IF Ausdruck [,] THEN {Befehl(e) / Zeilennummer}

[ELSE {Befehl / Zeilennummer}]

oder

IF Ausdruck GOTO Zeilennummer

[ELSE {Befehl(e) / Zeilennummer}]

Verwendung: Mit diesem Befehl wird eine Entscheidung über den Programmfluß je nach Ergebnis der Auswertung eines Ausdrucks gefällt.

Bemerkungen: Ergibt die Auswertung von "Ausdruck" einen WAHREN Wert, so wird die THEN-oder GOTO-Klausel ausgeführt. Auf THEN kann entweder eine Zeilennummer zur Verzweigung oder einer bzw. mehrere auszuführende Befehle folgen. Auf GOTO folgt stets eine Zeilennummer. Ist "Ausdruck" FALSCH, so wird die THEN- oder GOTO-Klausel ignoriert und die ELSE-Klausel, sofern vorhanden, ausgeführt. Ist kein ELSE vorhanden, so wird die Ausführung mit dem nächsten ausführbaren Befehl fortgesetzt. Vor THEN kann ein Komma stehen.

Beispiele:

```
10 INPUT "Wie ist die Außentemperatur in
   Grad Celsius";TEMP
20 IF TEMP <20 THEN PRINT "Es ist zu kalt
   für ein Picknick";GOTO 100
30 PRINT "Es ist warm genug für ein Picknick"
40 END 100 PRINT "Deshalb bleiben wir heute
   zuhause"
110 END
```

Beachten Sie, daß die Instruktion GOTO 100 nur ausgeführt wird, wenn TEMP <20 WAHR ist.

Zu derselben Entscheidung gelangt man auch folgendermaßen:

```
10 INPUT "Wie ist die Außentemperatur in
   Grad Celcius";TEMP
20 IF TEMP <20 THEN PRINT "Es ist zu kalt
   für ein Picknick":PRINT "Deshalb bleiben
   wir heute zuhause"ELSE PRINT"Es ist warm
   genug für ein Picknick".
```

In dem folgenden Beispiel werden Zeilennummern anstelle von Befehlen für denselben Zweck benutzt:

```
10 PRINT "Wie hoch ist die Außentemperatur in
   Grad Celsius";TEMP
20 IF TEMP <30 THEN 50
30 PRINT "Schalten Sie die Klimaanlage ein"
40 END
50 IF TEMP <20 THEN 100 ELSE 80
60 PRINT "GW-BASIC kommt nie zu dieser Zeile"
80 PRINT "Die Bedingungen sind gerade richtig
   für ein Picknick"
90 END
100 PRINT "Es ist noch zu kalt"
110 END
```

Verschachteln von IF...-Befehlen

IF...THEN...ELSE-Befehle können verschachtelt werden, d.h, die endgültige Aktion hängt von mehreren Entscheidungen ab.

Beispiele:

```
10 IF X>Y THEN PRINT "X IST GRÖSSER"
   ELSE IF Y>X THEN PRINT "X IST
   KLEINER"
   ELSE PRINT "GLEICH
```

Enthält der Befehl nicht dieselbe Anzahl von ELSE- und THEN-Klauseln, so wird jede ELSE- der nächsten THEN-Klausel zugeordnet, der noch kein Wert zugewiesen wurde. Nehmen Sie folgendes Beispiel:

```
10 INPUT "Wieviele Gramm Zucker";
   ZUCKER
20 INPUT "Wieviel Milch";
   MILCH
```

```
30 INPUT „Wieviele Pflaumen";
   OBST
40 IF ZUCKER>4 THEN IF MILCH=6 THEN
   100 ELSE IF OBST>10 THEN 120
50 PRINT "Die Mischung ist nicht süß genug.
   Oder es wurde nicht genügend Milch bzw.
   Pflaumen benutzt. Vielleicht ist auch alles
   falsch!"
60 END
100 PRINT "Diese Kuchenmischung hat wenig-
    stens genügend Zucker und die Milchmenge
    stimmt auch. Also mache ich mir keine
    Gedanken mehr um die Pflaumen"
110 END
120 PRINT "Es ist genügend Zucker vorhanden.
    Die Milchmenge stimmt nicht, aber es sind
    genügend Pflaumen vorhanden"
```

Ist nicht genügend Zucker vorhanden, so kann die Beurteilung von Zeile 120 nicht benutzt werden, selbst wenn genügend Pflaumen vorhanden sind.

Hinweis:

Folgt auf einen IF...THEN...-Befehl eine Zeilennummer im Direkt-Modus, so wird eine Fehlermeldung "Undefined line" (Undefinierte Zeile) angezeigt, es sei denn, vorher wurde im indirekten Modus ein Befehl mit der angegebenen Zeilennummer eingegeben.

Wird IF dazu benutzt, die Gleichheit eines Wertes zu überprüfen, der das Ergebnis einer Gleitpunkt-Berechnung ist, so muß daran gedacht werden, daß die interne Darstellung des Wertes unter Umständen nicht richtig ist. Deshalb sollte der Test in dem Bereich ausgeführt werden, in dem die Genauigkeit des Wertes schwanken kann. Um beispielsweise die berechnete Variable A mit dem Wert 1.0 zu vergleichen, wird folgender Befehl benutzt:

```
IF ABS(A-1.0)<1.0E-6 THEN...
```

Dieser Test ist wahr, wenn der Wert von A gleich 1.0 mit einem relativen Fehler von weniger als 1.0E-6 ist.

Der Wert 0 wird als "falsch" und ein von 0 abweichender Wert als "wahr" betrachtet. Deshalb könn-

ten Sie je nach Inhalt einer numerischen Variablen eine Entscheidung fällen. Beispiel:

```
210 IF IOFLAG THEN PRINT A$ ELSE
    LPRINT A$
```

Bei diesem Befehl wird die gedruckte Ausgabe je nach Wert der Variablen IOFLAG auf dem Bildschirm oder dem Zeilendrucker ausgegeben. Ist IOFLAG gleich Null, so geht die Ausgabe an den Zeilendrucker; ansonsten wird sie auf dem Bildschirm angezeigt.

INKEY$-Funktion

Syntax: INKEY$

Verwendung: Gibt ein Zeichen von dem Tastaturpuffer zurück.

Bemerkungen: Im Gegensatz zu dem INPUT-Befehl wartet INKEY$ nicht, bis Sie eine Taste betätigen. Versäumen Sie es, eine Taste zu betätigen, d.h., warten keine Zeichen in dem Tastaturpuffer, so gibt INKEY$ eine Nullzeichenfolge (Zeichenfolge mit der Länge 0) zurück. Ansonsten wird ein einzelnes Zeichen aus dem Tastaturpuffer zurückgegeben.

Durch eine Reihe von Tasten wird eine erweiterte Folge nicht aus einem, sondern aus zwei Zeichen zurückgegeben. In diesem Fall kann das erste Zeichen nicht angezeigt werden (ASCII-Code 0). Nur das zweite Zeichen ist bedeutungsvoll. Bei diesen Tasten handelt es sich um die Cursor-Tasten, sowie um bestimmte Kombinationen der Umschalt-, Ctrl- und Alt-Tasten (siehe Anhang B).

Ein weiterer Unterschied zwischen INKEY$ und INPUT besteht darin, daß INKEY$ die Tastatureingabe nicht auf dem Bildschirm wiedergibt.

Das von INKEY$ zurückgegebene Zeichen muß einer Zeichenfolgenvariablen zugewiesen werden, bevor es überprüft werden kann.

Beispiel: Das folgende Beispiel ist bei Programmen nützlich, bei denen der Benutzer beliebig Zeit zur Überprüfung des Bildschirminhaltes haben möchte. Die Programmausführung wird fortgesetzt, wenn eine beliebige Taste betätigt wird.

```
210 PRINT "Beliebige Taste betätigen, um fort-
    zufahren"
220 K$=INKEY$:IF LEN(K$)=0 THEN GOTO
    220
```

In dem nächsten Beispiel wird die Tastatureingabe fortlaufend dahingehend geprüft, ob Sie die Leertaste betätigt haben. Allerdings wird die Prüfung nicht unendlich fortgesetzt: nachdem INKEY$ die

Tastatureingabe so oft gelesen hat, wie mit LIMIT% angegeben, wartet das Programm nicht länger und teilt Ihnen mit einem akustischen Signal mit, daß Sie zu langsam sind. Ein derartiger Befehl könnte in Videospielen benutzt werden.

```
10 LIMIT% = 1000
20 FOR I% = 1 TO LIMIT%
30 K$ = INKEY$:IF K$ <>" " THEN 60
40 PRINT "Genau in der Zeit"
50 GOTO 20
60 NEXT I%
70 BEEP:CLS:PRINT "Zu langsam!"
80 GOTO 20
```

Um eine der erweiterten Zeichenfolgen zu überprüfen und zu lesen, wird folgender Befehl eingegeben:

```
10 K$ = INKEY$
20 IF LEN(K$)=2 THEN K$=RIGHT$(K$,1)
```

Mit INKEY$ werden die Sonderfunktionen der folgenden Tastenkombinationen nicht rückgängig gemacht:

<Ctrl-Break> (Unterbrechung des Programms)
<Ctrl-NumLock> (Systempause)
<PrtSc> (Druckt den Bildschirminhalt aus)
<Alt-Ctrl-Del> (Grundstellung)

INP-Funktion

Syntax: INP(I)

Verwendung: Rückgabe des von Anschlußposition I gelesenen Bytes. I muß in dem Bereich von 0 bis 65535 liegen.

Bemerkungen: INP ist die Zusatzfunktion zu dem OUT-Befehl.

Beispiel: 100 A% = INP(64)

Liest ein Byte von Anschlußposition 64 und weist es der Variablen A% zu.

Hinweis: INP führt dieselbe Funktion wie die IN-Instruktion in der Assemblersprache aus.

INPUT-Befehl

Syntax: INPUT [;]["<Eingabeaufforderung>";] Variable [,Variable ...]

Verwendung: Ermöglicht die Eingabe über die Tastatur während der Programmausführung.

Bemerkungen: Wird ein INPUT-Befehl angetroffen, so pausiert die Programmausführung. Mit einem auf dem Bildschirm angezeigten Fragezeichen wird angegeben, daß das Programm auf Daten wartet. Wird "Eingabeaufforderung" aufgenommen, so wird die entsprechende Zeichenfolge vor dem Fragezeichen ausgedruckt. GW-BASIC wartet dann, bis Sie Daten über die Tastatur eingeben. Die Dateneingabe muß durch Betätigung von <CR> abgeschlossen werden.

Hinter der Eingabeaufforderung kann ein Komma anstelle eines Semikolons benutzt werden, um das Fragezeichen zu unterdrücken. Durch INPUT "GEBURTSDATUM EINGEBEN", B$ wird die Eingabeaufforderung beispielsweise ohne Fragezeichen ausgedruckt.

Die eingegebenen Daten werden der bzw. den in "Variable" angegebenen Variable(n) zugewiesen. Die Anzahl von eingegebenen Datenelementen muß identisch mit der Anzahl von Variablen in der Liste sein. Vor jedem Datenelement, mit Ausnahme des ersten, muß ein Komma eingegeben werden.

Bei den Variablennamen in der Liste kann es sich um Namen von numerischen Variablen oder Zeichenfolgenvariablen (einschließlich indizierter Variablen) handeln. Der Typ jedes eingegebenen Datenelements muß mit dem von dem Variablennamen angegebenen Typ übereinstimmen. Eingegebene Zeichenfolgenelemente müssen nicht in Anführungszeichen stehen, es sei denn, sie enthalten Kommas oder beginnen bzw. enden mit einer signifikanten Anzahl von Leerstellen.

Wird INPUT mit zu wenig oder zu viel Elementen

oder mit dem falschen Variablentyp (numerische Variable anstelle einer Zeichenfolgenvariable usw.) beantwortet, so wird die Fehlermeldung "?Redo from start" (Von Anfang an neu beginnen) ausgedruckt. Eingabewerte werden erst dann wieder Variablen zugewiesen, wenn eine zulässige Antwort erteilt wurde. Wird nur eine einzige Variable in dem INPUT-Befehl angegeben, so kann das Datenelement weggelassen und einfach <CR> betätigt werden.

GW-BASIC liefert dann eine Null für eine numerische Variable oder eine leere Zeichenfolge für eine Zeichenfolgenvariable.

Beispiele:

```
10 INPUT X
20 PRINT X "IM QUADRAT" GLEICH; X^2
30 END
```

ergibt

```
? 5          (5 kann beispielsweise als Antwort auf
              das Fragezeichen eingegeben werden.)
5 IM QUADRAT GLEICH 25
```

```
10 PI=3.14
20 INPUT "WIE LAUTET DER RADIUS";R
30 A=PI*R^2
40 PRINT "DIE FLÄCHE DES KREISES IST
   GLEICH";A
50 PRINT
60 GOTO 20
```

ergibt

```
WIE LAUTET DER RADIUS? 7.4 (Sie geben 7.4
ein)
DIE FLÄCHE DES KREISES IST GLEICH
171.946

WIE LAUTET DER RADIUS? und so weiter
```

Um dieses Programm zu verlassen, benutzen Sie <Ctrl-Break>.

Hinweis:

Mit der Option >stdin, die in dem NCR-DOS Befehl für das Laden von GW-BASIC aufgenommen werden kann, kann eine Datei zugewiesen werden, die die Daten für INPUT enthält, so daß diese

nicht über die Tastatur eingegeben zu werden brauchen. In diesem Fall muß jede Antwort mit <Ctrl-Z> abgeschlossen werden. Wird dieses Zeichen nicht angegeben, so wird eine Fehlermeldung "Read past end" (Über Ende hinausgelesen) angezeigt. Sämtliche Dateien werden abgeschlossen und GW-BASIC gibt die Kontrolle zur Befehlsebene von NCR-DOS zurück (nicht "Ok"). Bei Betätigung von <Ctrl-Break> wird ebenfalls auf NCR-DOS Befehlsebene zurückgegangen.

Wird <Ctrl-Break> benutzt, um die Ausführung eines INPUT-Befehls zu unterbrechen, und wird danach mit CONT zu dem Programm zurückgekehrt, so nimmt GW-BASIC die Ausführung mit dem INPUT-Befehl (und nicht mit dem folgenden Befehl) wieder auf.

In dem folgenden Beispiel wird davon ausgegangen, daß eine Datei mit Informationen über Telefonteilnehmer (Name, Nummer) in Datensätzen mit einer Länge von jeweils 35 Bytes vorhanden ist. Mit der ersten Zeile wird die Datei für den Direktzugriff eröffnet.

```
10 OPEN "R",#1, "TELNUMS",35
```

Der erste Datensatz enthält nur die Anzahl von Teilnehmern bis zu einer Höchstzahl von 99. Für diesen ersten Datensatz wäre folgende FIELD-Definition möglich:

```
20 FIELD 1,2 AS SUBSCRIB$,33 AS UNUSED$
```

Die restlichen Sätze mit den tatsächlichen Namen und Telefonnummern könnten folgendermaßen aufgeteilt sein.

```
30 FIELD 1,25 AS NNAME$,10 AS
   PHONENO$
```

Als erstes wird der erste Datensatz mit GET geholt. Danach werden die beiden ersten Bytes dieses Datensatzes überprüft, um zu sehen, wieviele Teilnehmerdatensätze in der Datei stehen. (Diese Zahl wird aus einer Zeichenfolge in einen ganzzahligen Wert umgewandelt.):

```
40 GET #1
50 TOTAL% = CVI(SUBSCRIB$)
```

Mit dem restlichen Programm wird jeder Datensatz mit GET aus der Plattendatei geholt. Danach werden jeder Name und jede Telefonnummer auf dem Bildschirm angezeigt:

```
60 FOR LOOP% = 2 TO TOTAL%
70 GET #1, LOOP%
80 PRINT NNAME$,PHONENO$
90 NEXT LOOP%
100 END
```

INPUT#-Befehl

Syntax: INPUT#<Dateinummer>,<Variablenliste>

Verwendung: Liest Datenelemente aus einer sequentiellen Einheit oder Datei und weist sie Programmvariablen zu.

Bemerkungen: <Dateinummer> entspricht der Nummer, die beim Eröffnen der Datei (mit OPEN) zur Eingabe benutzt wurde. "Variablenliste" enthält die Variablennamen, denen Elemente in der Datei zugewiesen werden. (Der Variablentyp muß mit dem von dem Variablennamen angegebenen Typ übereinstimmen.) Bei INPUT# wird kein Fragezeichen ausgedruckt wie bei INPUT.

Die Datenelemente in der Datei müssen genau so angezeigt werden, als würden die Daten als Antwort auf einen INPUT-Befehl eingegeben. Bei numerischen Werten werden führende Leerzeichen, Zeilenschaltungen und Zeilenvorschübe ignoriert. Das erste Zeichen, bei dem es sich nicht um ein Leerzeichen, eine Zeilenschaltung oder einen Zeilenvorschub handelt, wird als Anfang der Zahl betrachtet. Die Zahl wird mit einem Leerzeichen, einer Zeilenschaltung, einem Zeilenvorschub oder einem Komma beendet.

Sucht GW-BASIC die sequentielle Datendatei nach einem Zeichenfolgenelement ab, so werden führende Leerzeichen, Zeilenschaltungen und Zeilenvorschübe ebenfalls ignoriert. Das erste Zeichen, bei dem es sich nicht um ein Leerzeichen, eine Zeilenschaltung oder einen Zeilenvorschub handelt, wird als Beginn des Zeichenfolgenelementes angesehen. Handelt es sich bei diesem ersten Zeichen um ein Anführungszeichen ("), so besteht das Zeichenfolgenelement aus sämtlichen Zeichen, die zwischen dem ersten und zweiten Anführungszeichen gelesen werden. Deshalb darf eine in Anführungszeichen stehende Zeichenfolge kein Anführungszeichen als Zeichen enthalten. Ist das erste Zeichen der Zeichenfolge kein Anführungszeichen, so handelt es sich um eine nicht in Anfüh-

rungszeichen stehende Zeichenfolge, die mit einem Komma, einer Zeilenschaltung oder einem Zeilenvorschub (oder nachdem 255 Zeichen gelesen wurden) beendet wird. Wird während der Eingabe eines numerischen Elementes oder Zeichenfolgenelementes (mit INPUT) ein Dateiende erreicht, so wird das Element beendet.

Beispiel: Siehe Kapitel 5 "Dateien und Einheiten".

Hinweis: INPUT# kann auch mit einer Direktzugriffsdatei benutzt werden.

INPUT$-Funktion

Syntax: INPUT$(X[,[#]Y])

Verwendung: Rückgabe einer Folge von X Zeichen, die aus der Datei mit der Nummer Y gelesen werden. Wird die Dateinummer nicht angegeben, so werden die Zeichen über die Tastatur eingelesen.

Bemerkungen: Wird die Eingabe über die Tastatur vorgenommen, so werden keine Zeichen auf dem Bildschirm im Echoverfahren wiedergegeben. Sämtliche Steuerzeichen werden direkt übergeben. Dies gilt jedoch nicht für <Ctrl-Break>, das zur Unterbrechung der Ausführung der INPUT#-Funktion benutzt werden kann. Wird INPUT# über die Tastatur beantwortet, so braucht <CR> nicht betätigt zu werden. Durch X wird die Anzahl von eingegebenen Zeichen ohnehin begrenzt.

Beispiel:

```
5 'INHALT EINER SEQUENTIELLEN DATEI
  IN HEXADEZIMALFORM AUFLISTEN
10 OPEN "I",1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Hinweis: Mit INPUT$ und INKEY$ werden sämtliche Tastatureingaben und nicht nur die ausdruckbaren Zeichen gelesen. Soll beispielsweise die Betätigung von Cursor-Tasten ermittelt werden, so werden diese Funktionen und nicht INPUT oder LINE INPUT benutzt.

Gleichermaßen sollten Datenübertragungsdateien mit INPUT$ (und nicht mit INPUT# oder LINE INPUT#) gelesen werden, da jedes ASCII-Zeichen signifikant sein kann.

INSTR-Funktion

Syntax: INSTR([I,]X$,Y$)

Verwendung: Sucht nach dem ersten Auftreten der Zeichenfolge Y$ in X$ und gibt die Position zurück, an der die Übereinstimmung gefunden wurde. Mit dem wahlweisen Offset I wird die Position in X$ festgelegt, ab der mit der Suche begonnen wird.

Bemerkungen: I muß in dem Bereich von 1 bis 255 liegen. Ist I größer als die Anzahl von Zeichen in X$, ist X$ gleich 0 oder kann Y$ nicht gefunden werden, so gibt INSTR Null zurück. Ist Y$ gleich Null, so gibt INSTR I zurück. Wurde I nicht angegeben, so wird 1 zurückgegeben. Bei X$ und Y$ kann es sich um Zeichenfolgenvariablen, Zeichenfolgenausdrücke oder Zeichenfolgenkonstanten handeln.

Beispiel:

```
10 X$ = "ABCDEB"
20 Y$ = "B"
30 PRINT INSTR(X$,Y$);INSTR(4,X$,Y$)
ergibt
2   6
```

Die erste INSTR-Funktion sucht ab Anfang der Zeichenfolgenvariablen X$ nach "B". In Position 2 wird ein "B" gefunden. Die zweite INSTR-Funktion beginnt ab Position 4 mit der Suche und kann demzufolge das "B" in Position 2 nicht finden, findet jedoch das "B" in Position 6.

Hinweis: Liegt I außerhalb des zulässigen Bereichs, d.h. außerhalb der Länge von X$, so wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt. Die Länge einer Zeichenfolge kann mit der LEN-Funktion ermittelt werden.

INT-Funktion

Syntax: INT(X)

Verwendung: Gibt die größte Ganzzahl zurück, die kleiner als oder gleich dem numerischen Wert X ist.

Beispiele:

```
PRINT INT(99.89)
ergibt
 99

PRINT INT(-12.11)
ergibt
 -13
```

Hinweis: Hier wird auf die Funktionen CINT und FIX verwiesen, die ebenfalls Ganzzahlwerte zurückgeben.

KEY-Befehl

Syntax:

KEY <Tastennummer>,<Zeichenfolgenausdruck>
KEY ON
KEY OFF
KEY LIST

<Tastennummer>
gibt die Nummer einer programmierbaren Funktionstaste in dem Bereich von 1 bis 10 an (siehe nachfolgende Liste).

<Zeichenfolgenausdruck>
gibt den Zeichenfolgenausdruck an, der der programmierbaren Funktionstaste zugewiesen wird.

Verwendung:

Mit diesem Befehl können Sie den programmierbaren Funktionstasten einen Zeichenfolgenausdruck zuweisen. Jeder dieser Tasten kann eine Zeichenfolge aus maximal 15 Zeichen zugewiesen werden. Wird die Taste dann später betätigt, so wird die entsprechende Zeichenfolge in GW-BASIC eingegeben. Mit KEY ON/OFF/LIST kann der Inhalt der programmierbaren Funktionstasten angezeigt oder versteckt werden.

Bemerkungen:

Ursprünglich werden den programmierbaren Funktionstasten die folgenden Werte von GW-BASIC zugewiesen:

F1 LIST	F6, "LPT1:<CR>
F2 RUN<CR>	F7 TRON<CR>
F3 LOAD"	F8 TROFF<CR>
F4 SAVE"	F9 KEY
F5 CONT<CR>	F10 SCREEN 0,0,0<CR>

KEY <Tastennummer>, <Zeichenfolgenausdruck>
Weist den Zeichenfolgenausdruck der angegebenen Taste zu. Der Zeichenfolgenausdruck kann eine Länge von 1 bis 15 Zeichen aufweisen. Ist er länger als 15 Zeichen, so werden nur die ersten 15 Zeichen zugewiesen.

Wird für <Tastennummer> ein Wert angegeben, der nicht in dem Bereich von 1 bis 10 liegt, so wird die Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt. Die der Taste zuvor zugeordnete Zeichenfolge wird beibehalten.

Wird einer programmierbaren Funktionstaste eine Zeichenfolge mit der Länge 0 zugewiesen, so wird die Taste deaktiviert. Sie bleibt deaktiviert, bis ihr ein anderer Zeichenfolgenausdruck zugewiesen wird.

KEY ON
Die sechs ersten Zeichen jeder Funktionstaste werden auf der 25sten Zeile des Bildschirms angezeigt, genau wie nach dem Laden von GW-BASIC. Wird eine Anzeigebreite von 40 Spalten benutzt, so werden nur fünf Funktionstasten angezeigt.

KEY OFF
Löscht die Anzeige der programmierbaren Funktionstasten aus der 25sten Zeile, deaktiviert die Funktionstasten jedoch nicht.

KEY LIST
Listet die Werte aller zehn programmierbaren Funktionstasten auf dem Bildschirm auf. Alle 15 Zeichen jeder Funktionstaste werden angezeigt.

Wird eine programmierbare Funktionstaste zugewiesen, so gibt die INKEY$-Funktion bei jedem Aufruf ein Zeichen der Zeichenfolge zurück. Wird die programmierbare Funktionstaste deaktiviert, so gibt INKEY$ eine aus zwei Zeichen bestehende Zeichenfolge zurück. Das erste Zeichen ist eine binäre Null, während das zweite Zeichen den Tasten-Suchcode darstellt (siehe Anhang B).

Beispiele:

In dem folgenden Beispiel weist der Befehl in Zeile 10 der Funktionstaste 1 die Zeichenfolge 'MENU' mit abschließendem <CR> zu. Mit dieser Zuweisung kann ein Menü auf dem Bildschirm angezeigt werden, wenn der Benutzer in dieses Menü geht. Mit Zeile 20 wird die Taste deaktiviert.

```
10 KEY 1, "MENU" + CHR$(13)
20 KEY 1,""
```

Programmverzweigung bei Tastenbetätigung

Bei GW-BASIC können sechs zusätzliche Programmverzweigungen bei Tastenbetätigung definiert werden. Die unterbrochene Taste muß sich im Ctrl-, Shift- oder Alt-Modus befinden.

Um eine Programmverzweigung festzulegen, ist ein KEY-Befehl mit folgenden Elementen erforderlich:

KEY Tastennummer,CHR$(Modus) + CHR$(Tastatur)

<Tastennummer> ist eine Zahl in dem Bereich von 15 bis 20. „Modus" stellt einen der folgenden Hexadezimalwerte dar:

Caps Lock	&H40
Num Lock	&H20
Alt	&H08
Ctrl	&H04
Shift	&H01 oder &H02 oder &H03

Kombinierte Tasten-Aktionen werden durch Kombination der entsprechenden Codes erzielt. So bedeutet &H04 + &H01 beispielsweise <Ctrl-Shift>.

"Tastatur" ist eine Zahl, mit der die Position der zu unterbrechenden Taste auf der Tastatur dargestellt wird (siehe Anhang F).

Beispiel:

Mit dem folgenden Programm wird eine Programmverzweigung für die Tastenkombination <Ctrl-Shift-X> festgelegt:

```
10 KEY 15,CHR$(&H04+&H03)+CHR$(45)
20 ON KEY(15) GOSUB 1000
30 KEY (15) ON
 .
 .
 .
1000 PRINT "Jemand hat Ctrl-Shift-X betätigt!"
 .
```

In Zeile 20 wird angegeben, wann diese Tastenkombination betätigt wird. Vorausgesetzt die Programmverzweisung ist aktiviert (Zeile 30, siehe KEY(n)-Befehl), verzweigt sich GW-BASIC bei Zeile 1000 zu der Unterroutine.

Hinweis:

Mit der Programmverzweigung bei Tastenbetätigung kann die Funktion Ctrl-PrtSc nicht außer Kraft gesetzt werden. Außerdem wirkt sie sich nicht auf die Funktions- oder Cursor-Tasten aus.

Sie selbst können jedoch die Ctrl-Break-Funktion von GW-BASIC außer Kraft setzen. In diesem Fall können Sie diese Tastenkombination nicht mehr dazu benutzen, aus einem programmierten Rücksprung zu der "Ok"-Ebene von GW-BASIC auszubrechen. Gleichermaßen können Sie die Tastenkombination Ctrl-Alt-Del (Tastenkombination zur Grundstellung) außer Kraft setzen.

KEY(N)-Befehl

Syntax:

KEY(n) ON
KEY(n) OFF
KEY(n) STOP

Wobei (n) die Nummer einer programmierbaren Funktionstaste, einer Cursor-Taste oder einer der sechs vom Benutzer definierten Tasten-Unterbrechungen darstellt (siehe KEY-Befehl).

Verwendung:

Mit diesem Befehl wird die Programmverzweigung bei Betätigung einer der oben angeführten Tasten oder Tastenkombinationen aktiviert bzw. deaktiviert.

Bemerkungen:

n ist ein numerischer Ausdruck in dem Bereich von 1 bis 20:

1 – 10	Entsprechende Funktionstaste
11	Cursor nach oben
12	Cursor nach links
13	Cursor nach rechts
14	Cursor nach unten
15 – 20	Vom Benutzer definierte, unterbrechbare Tastenkombination zur Programmverzweigung

Mit dem Befehl KEY(n) ON wird die Programmverzweigung bei Betätigung der von n angegebenen Taste bzw. Tastenkombination aktiviert. Ist die Programmverzweigung aktiviert und wird eine von Null abweichende Zeilennummer in dem ON KEY Befehl angegeben, so prüft GW-BASIC vor Ausführung jedes Befehls, ob die angegebene Taste benutzt wurde. Wenn ja, nimmt GW-BASIC eine Verzweigung zu der Subroutine vor, wobei mit der in dem ON KEY Befehl angegebenen Zeile begonnen wird.

Mit KEY(n) OFF wird die Programmverzweigung bei einem Ereignis deaktiviert. Tritt ein Ereignis ein, so wird es nicht festgehalten.

Mit KEY(n) STOP wird die Unterbrechung bei einem Ereignis deaktiviert. Tritt jedoch ein Ereig-

nis ein, so wird es festgehalten und der ON KEY-Befehl wird ausgeführt, sobald die Programmverzweigung aktiviert wird.

Beispiel: Siehe KEY.

Hinweis: Ein KEY(n)-Befehl darf nicht vor einem ON KEY(n)-Befehl stehen.

KEY(n) ON hat keine Auswirkung auf die Anzeige der Funktionstaste in der 25sten Bildschirmzeile.

KILL-Befehl

Syntax: KILL<Dateispez>

Verwendung: Löscht eine Datei von einer Diskette.

Bemerkungen: Wird ein KILL-Befehl für eine Datei angegeben, die gerade eröffnet ist, so wird eine Fehlermeldung "File already open" (Datei schon eröffnet) angezeigt.

KILL kann für sämtliche Plattendateien benutzt werden, d.h. sowohl für Programm- als auch für Datendateien. Die Dateispezifikation kann Fragezeichen (?) oder Sternchen (*) (Dateigruppensymbole) enthalten. Ein Fragezeichen entspricht jedem einzelnen Zeichen in dem Dateinamen oder der Dateinamenerweiterung. Ein Sternchen entspricht einem oder mehreren Zeichen ab der Position, an der das Sternchen angegeben wird.

Beispiele:

```
200 KILL "A:OBSOLETE.DAT"
```

löscht eine Datei namens OBSOLETE.DAT in Laufwerk A.

```
200 KILL
"LOCAL1 \LOCAL1A \UNUSED.BAS"
```

löscht die Datei UNUSED.BAS aus dem Unterverzeichnis LOCAL1A auf der aktuellen Diskette.

Hinweis: KILL ist gleichbedeutend mit dem NCR-DOS Befehl ERASE.

LCOPY-Befehl

Syntax: LCOPY

Verwendung: Zur Ausgabe der Bildschirmanzeige an den Drukker.

Bemerkung: Der Befehl LCOPY ermöglicht es Ihnen einen Bildschirm – "Dump" auszuführen, d. h., den Bildschirminhalt als exaktes Abbild auf dem Drucker zu erstellen.

Die für das Drucken benötigte Zeit hängt von der von Ihnen gewählten Bildschirmauflösung ab.

Zum Ausdrucken von Bildschirmgrafiken auf einem Drucker kann auch der GRAPHICS-Befehl verwendet werden. Einzelheiten über den Befehl GRAPHICS können Sie dem NCR-DOS-Handbuch entnehmen.

LEFT$ Funktion

Syntax: LEFT$(X$,I)

Verwendung: Gibt eine Zeichenfolge einschließlich der linksbündigsten I Zeichen von X$ zurück.

Bemerkungen: I muß in dem Bereich von 0 bis 255 liegen. Ist I größer als die Anzahl von Zeichen in X$, so wird die gesamte Zeichenfolge (X$) zurückgegeben. Ist I = 0, so wird die Nullfolge (Zeichenfolge mit der Länge 0) zurückgegeben.

Beispiel:

```
10 A$ = "BASIC"
20 B$ = LEFT$(A$,3)
30 PRINT B$
```

ergibt

```
BAS
```

Siehe auch die Funktionen MID$ und RIGHT$.

Hinweis: Die Länge einer Zeichenfolge kann mit der LEN-Funktion ermittelt werden.

LEN-Funktion

Syntax: LEN(X$)

Verwendung: Gibt die Anzahl von Zeichen in X$ zurück, einschließlich Leerzeichen und Sonderzeichen.

Beispiel:
```
10 X$ = "PORTLAND, OREGON"
20 PRINT LEN(X$)
ergibt
 16
```

LET-Befehl

Syntax: [LET] <Variable>=<Ausdruck>

Verwendung: Weist einer Variablen den Wert eines Ausdrucks zu.

Bemerkungen: Beachten Sie, daß das Wort LET wahlweise ist; d.h., das Gleichzeichen reicht für die Zuweisung eines Ausdrucks an einen Variablennamen aus.

Beispiel:

```
110 LET D=12
120 LET E=12^2
130 LET F=12^4
140 LET SUM=D+E+F
150 LET TEXT$="Wörter"
 .
 .
 .
```

ist gleichbedeutend mit

```
110 D=12
120 E=12^2
130 F=12^4
140 SUM=D+E+F
150 TEXT$="Wörter"
 .
 .
 .
```

Hinweis: Dieselbe Variable kann rechts und links von dem Gleichzeichen stehen. Zum Beispiel:

```
110 INPUT "Beliebige Zahl eingeben", NUMB
110 NUMB = NUMB/3
120 PRINT "Ihre Zahl dividiert durch drei =
    ";NUMB
```

Ergibt der Ausdruck rechts von dem Gleichzeichen keinen Wert desselben Typs wie bei der Variablen links von dem Gleichzeichen, so wird eine Fehlermeldung "Type mismatch" (Übereinstimmungsfehler) angezeigt.

LINE-Befehl

Syntax: LINE[(x1,y1)]-(x2,y2)[,[Farbe][,B[F]][,Linienart]]

Verwendung: Zeichnet eine Linie, ein Kästchen oder ein ausgefülltes Kästchen auf dem Bildschirm.

(x1,y1),(x2,y2)
gibt die Koordinaten in absoluter oder verschobener Form an (siehe Kapitel 3 "Bildschirmanzeige"). Werden die Punktkoordinaten (x1,y1) nicht angegeben, so entspricht der Anfangspunkt der Linie dem letzten Punkt, auf den in einem Grafik-Befehl Bezug genommen wurde.

"Farbe"
gibt die Farbe der Linie, des Kästchens oder ausgefüllten Kästchens an. Bei Farbgrafiken mit niedriger und hoher Auflösung muß die Zahl in dem Bereich von 1 bis 3 liegen, um eine Farbe aus der Farbpalette anzugeben. Die Zahl kann jedoch auch 0 (Hintergrundfarbe) sein. Der Standardwert lautet 3. Bei einfarbiger Grafik mittlerer und hoher Auflösung gibt 0 schwarz und 1 weiß an. Der Standardwert ist 1.

B oder BF
gibt ein Kästchen oder ausgefülltes Kästchen an. Mit B wird BASIC angewiesen, ein Rechteck zu zeichnen, bei dem die Punkte (x1,y1) und (x2,y2) die gegenüberliegenden Ecken darstellen. Dadurch brauchen keine vier LINE-Befehle zur Ausführung derselben Funktion angegeben zu werden:

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

Mit BF wird BASIC angewiesen, dasselbe Rechteck wie bei B zu zeichnen und die Innenpunkte in derselben Farbe wie bei B auszufüllen. (Mit B kann "Linienart" nicht benutzt werden.)

"Linienart" ist eine Option, bei der die Linie oder das Kästchen nicht mit einer fortlaufenden Linie,

sondern gepunktet oder gestrichelt bzw. in einem anderen von Ihnen ausgewählten Muster gezeichnet wird. "Linienart" ist eine Zahl in dem Bereich von 0 bis 65535. Die Linienart der Zeichnung entspricht der Binärdarstellung der Zahl mit 16 Bits. Eine gepunktete Linie setzt das Bit-Muster

1010101010101010 oder
0101010101010101

voraus. Dies entspricht der Dezimalzahl 43690 (erstes Muster) oder 21845 (zweites Muster). Erfahrene Programmierer bevorzugen wahrscheinlich die Hexadezimalschreibweise &HAAAA und &H5555.

Das durch Linienart festgelegte Muster gilt für die 16 ersten Bildschirmpunkte der Linie und wird danach bis zum Ende der Linie wiederholt. Nicht festgelegte Bildschirmpunkte behalten das alte Erscheinungsbild bei. Bevor eine unterbrochene Linie gezeichnet wird, wird deshalb empfohlen, zuerst eine durchgezogene Linie mit der Hintergrundfarbe (0) zu zeichnen, um eine einzelne Hintergrundfarbe für die gesamte Länge der Zeile zu erhalten. Dies ist eine Überlegung, die insbesondere bei Farbgrafiken berücksichtigt werden sollte.

Beispiele:

```
10 INPUT "X2 und Y2 und Farbe";
   X2%, Y2%, COL%
20 SCREEN 1.0:COLOR 0,1
30 FOR D% = 1 TO 400:NEXT D%
40 LINE - (X2%,Y2%),COL%
50 IF INKEY$ = "" THEN 40
60 SCREEN 0:WIDTH 80
```

Mit diesem Programm werden Sie um Angabe der End-Koordinaten und der Farbe einer Linie gebeten, die von der Mitte des Bildschirms gezeichnet werden soll (dem Punkt, auf den als letztes im Anschluß an den SCREEN1-Befehl Bezug genommen wurde.) Danach wird die Linie gezeichnet und bleibt auf dem Bildschirm stehen, bis Sie eine beliebige Taste betätigen. (Die kurze Verzögerung durch Zeile 30 wird nur zur Beruhigung der Bildschirmanzeige nach Änderung der Betriebsart benutzt.)

In dem folgenden Beispiel werden die Ecken des Bildschirms durch eine Linie miteinander verbunden, die mit mittlerer Auflösung gezeichnet wird.

```
10 SCREEN 2:CLS
20 LINE (0,0)-(639,199),,B
30 LINE (0,0)-(639,199)
40 LINE (0,199)-(639,0)
50 IF INKEY$ = ""THEN 50
60 SCREEN 0
```

Da "Linienart" nicht angegeben wird, werden fortlaufende Linien gezeichnet. Nun werden die Zeilen 20 bis 40 folgendermaßen neu geschrieben:

```
20 STYLE%=21845:LINE
   (0,0)-(639,199),,B,STYLE%
30 LINE (0,0)-(639,199),,,STYLE%
40 LINE (0,199)-(639,0),,,STYLE%
```

Dies führt zu gepunkteten Linien. Der Wert von STYLE% wird auf 1 geändert. Nun werden weniger Punkte in weitaus größerem Abstand gezeichnet. Der Wert 3855 erzeugt gestrichelte Linien. Das Binäräquivalent lautet:

0000111100001111 (Hexadezimalwert 0F0F)

In dem folgenden Beispiel wird eine Linie mit verschobenen Koordinaten gezeichnet, d.h. relativ zu dem letzten Punkt, auf den Bezug genommen wurde (in diesem Fall 150,100):

```
10 LINE (150,100)-STEP (30,-30)
```

Mit dem folgenden Beispiel werden Kästchen mit beliebiger Größe mit maximal 50 X 50 Punkten an beliebigen Stellen in beliebigen Farben gezeichnet:

```
10 SCREEN 1,0
20 COLOR 0,RND*4
30 LINE
(RND*319,RND*199)-STEP(RND*50,RND*50)
,RND*3,BF
40 GOTO 20
```

Hinweis:

Versucht LINE über die Grenzen des Grafik-Anzeigebereichs hinauszuzeichnen, so kommt es zu keiner Umschaltung zu einem anderen Teil des Bildschirms. Die Teile der Zeichnung, die außerhalb der Grenzen liegen, fallen einfach weg. GW-BASIC betrachtet dies nicht als eine Fehlerbedingung.

LINE INPUT-Befehl

Syntax: LINE INPUT[;][<Eingabeaufforderung>;] <Zeichenfolgenvariable>

Verwendung: Liest eine ganze Zeile (bis zu 254 Zeichen) von der Tastatur in eine Zeichenfolgenvariable, wobei Abgrenzungszeichen ignoriert werden.

Bemerkungen: <Eingabeaufforderung> ist eine Zeichenfolgenkonstante, die auf dem Bildschirm angezeigt wird, bevor die Eingabe akzeptiert wird. Ein Fragezeichen wird nicht angezeigt, es sei denn, es ist Teil der <Eingabeaufforderung>. Die nachfolgende Tastatureingabe wird <Zeichenfolgenvariable> zugewiesen.

Folgt unmittelbar auf LINE INPUT ein Semikolon, so bleibt der Cursor auf derselben Zeile stehen, selbst wenn Sie <CR> betätigt haben.

Ein LINE INPUT Befehl kann durch Eingabe von <Ctrl-Break> abgebrochen werden. Danach kehrt GW-BASIC auf Befehlsebene ("Ok") zurück.

Beispiel: Siehe "LINE INPUT#"

Hinweis: Wird die Ausführung eines LINE INPUT Befehls durch Betätigung von <Ctrl-Break> unterbrochen und wird dann mit CONT zu dem Programm zurückgegangen, so nimmt GW-BASIC die Ausführung mit dem LINE INPUT Befehl (und nicht mit dem nachfolgenden Befehl) wieder auf.

LINE-INPUT#-Befehl

Syntax: LINE INPUT# <Dateinummer>,<Zeichenfolgenvariable>

Verwendung: Liest eine ganze Zeile (bis zu 254 Zeichen) von einer sequentiellen Datendatei auf Platte in eine Zeichenfolgenvariable, wobei Abgrenzungszeichen ignoriert werden.

Bemerkungen: <Dateinummer> ist die Nummer, unter der die Datei mit dem OPEN-Befehl eröffnet wurde. <Zeichenfolgenvariable> entspricht dem Variablennamen, dem die Zeile zugewiesen wird. Mit LINE INPUT# werden sämtliche Zeichen in der sequentiellen Datei bis zu einer Zeilenschaltung gelesen. Die Zeilenschaltung/der Zeilenvorschub wird dann übersprungen. Der nächste LINE INPUT# Befehl liest sämtliche Zeichen bis zu der nächsten Zeilenschaltung. (Wird eine Folge "Zeilenvorschub/Zeilenschaltung" angetroffen, so wird sie beibehalten.)

LINE INPUT# ist besonders nützlich, wenn jede Zeile einer Datendatei in Felder unterteilt ist, oder wenn ein im ASCII-Format gesichertes GW-BASIC Programm (siehe SAVE) von einem anderen Programm in Form von Daten gelesen wird.

Beispiel:

```
10 OPEN "O",1,"LISTE"
20 LINE INPUT "KUNDEN
INFORMATION? ";C$
30 PRINT #1,C$
40 CLOSE 1
50 OPEN "I",1,"LISTE"
60 LINE INPUT #1,C$
70 PRINT C$
80 CLOSE 1
ergibt
KUNDEN INFORMATION? LINDA
JONES   234,4   MEMPHIS
LINDA JONES   234,4   MEMPHIS
```

Bei diesem Beispiel werden Informationen mit Kommas und anderen Abgrenzungszeichen über die Tastatur in C$ gelesen. Danach wird die sequentielle Datei wieder eröffnet, die Informationen werden wieder in C$ eingelesen und danach angezeigt.

LIST-Befehl

Syntax: LIST [<Zeilennummer 1>][-[Zeilennummer 2]] [<Dateispez>]

<Zeilennummer 1>, <Zeilennummer 2>
sind Zahlen in dem Bereich von 0 bis 65529. Mit ihnen wird der Bereich der Programmzeilen angegeben, die angezeigt werden sollen.

<Dateispez>
Eine Datei oder Einheit, zu der die Programmauflistung geleitet werden soll. Wird die Dateispezifikation weggelassen, so werden die Programmzeilen auf dem Bildschirm aufgelistet.

Verwendung: Ermöglicht die Auflistung eines Programms, im allgemeinen auf dem Bildschirm.

Bemerkungen: Werden keine Zeilennummern angegeben, so wird das gesamte Programm aufgelistet.

Auflistungen auf dem Bildschirm können durch Betätigung von <Ctrl-Break> beendet werden. Mit <Ctrl-Num Lock> wird die Auflistung vorübergehend eingestellt, bis Sie eine beliebige Taste betätigen.

Beispiele: LIST

Zeigt das gesamte Programm an.

LIST 100

Zeigt nur Zeile 100 an.

LIST 80-

Mit dem Bindestrich wird angegeben, daß nicht nur Zeile 80, sondern auch jede nachfolgende Zeile mit einer höheren Zeilennummer angezeigt wird.

LIST-120

Jede Zeile bis einschließlich Zeile 120 wird angezeigt.

LIST 50-210

Hier ist die Auflistung auf den angegebenen Zeilenbereich beschränkt.

LIST 1000-"COPY2".BAS

Schreibt eine Kopie des Programms im ASCII-Format (siehe SAVE) in die Datei COPY2.BAS, wobei mit Zeile 1000 begonnen wird.

Hinweis:

Wird LIST im indirekten Modus ausgeführt, so kehrt GW-BASIC unmittelbar danach zu der Befehlsebene ("Ok") zurück.

Für weitere Informationen über die Benutzung von Einheitennamen (z.B. LPT1:) für die Dateispezifikation wird auf Kapitel 5 "Dateien und Einheiten" verwiesen.

LLIST-Befehl

Verwendung: LLIST[<Zeilennummer 1>]
[-[<Zeilennummer 2>]]

Zweck: Listet ein Programm auf dem Drucker auf.

Bemerkungen: LLIST wird wie LIST ausgeführt, nur daß keine <Dateispez> benutzt werden kann.

LOAD-Befehl

Syntax: LOAD <Dateispez> [,R]

Verwendung: Lädt eine Datei von einer Diskette oder einer Einheit in den Speicher.

Bemerkungen: Die <Dateispez> muß den Dateinamen enthalten, der beim Sichern der Datei mit SAVE benutzt wurde. Die Erweiterung braucht jedoch nicht angegeben zu werden, wenn sie .BAS lautet.

Wird die Option R angegeben, so wird das Programm automatisch ausgeführt, nachdem es geladen wurde.

LOAD schließt sämtliche eröffneten Dateien ab und löscht alle Variablen und Programmzeilen, die gerade im Speicher stehen, bevor das angegebene Programm geladen wird. Wird jedoch die Option R mit LOAD benutzt, so wird das Programm nach dem Laden ausgeführt. Sämtliche eröffneten Datendateien bleiben geöffnet, damit das neu ausgeführte Programm sie benutzen kann.

Beispiel: LOAD "STRTRK",R

Mit diesem Befehl wird das Programm STRTRK.BAS geladen und ausgeführt. (Hat dieselbe Auswirkung wie RUN "STRTRK".)

LOAD "B:MYPROG"

lädt das Programm MYPROG.BAS von der Diskette in Laufwerk B, führt das Programm jedoch nicht aus.

Hinweis: <Dateispez> kann einen Pfadnamen enthalten.

LOC-Funktion

Syntax: LOC(<Dateinnummer>)

wobei <Dateinummer> die Nummer darstellt, unter der die Datei eröffnet wurde.

Verwendung: Bei Direktzugriffsdateien gibt LOC die Nummer des letzten gelesenen oder geschriebenen Satzes zurück.

Bei Dateien mit sequentiellem Zugriff gibt LOC die Anzahl von Sätzen an, die seit dem Eröffnen der Datei von der Datei gelesen oder in die Datei geschrieben wurden.

Bemerkungen: Wird eine Datei für die sequentielle Eingabe eröffnet, so liest GW-BASIC den ersten Sektor der Datei, so daß LOC eine 1 zurückgibt, noch bevor es zu einer Eingabe von der Datei kommt.

Bei einer Datenübertragungsdatei wird mit LOC ermittelt, ob Zeichen in der Eingabewarteschlange auf das Lesen warten. Stehen mehr als 255 Zeichen in der Warteschlange, so gibt LOC 255 zurück. Da Interpreter-Zeichenfolgen auf 255 Zeichen beschränkt sind, ist es vor dem Einlesen von Daten nicht mehr erforderlich, die Zeichenfolgengröße zu testen. Stehen weniger als 255 Zeichen in der Warteschlange, so entspricht der von LOC zurückgegebene Wert der tatsächlichen Anzahl von Zeichen, die auf das Einlesen warten.

Beispiele: 200 IF LOC(1)>50THEN STOP

stoppt das Programm, wenn über den 50sten Datensatz hinaus gelesen wurde.

Das folgende Beispiel wird besonders dann benutzt, wenn ein Satz einer Direktzugriffsdatei, der gerade gelesen wurde, erneut geschrieben werden soll:

```
200 PUT #1,LOC(1).
```

LOCATE-Befehl

Syntax: LOCATE [<Zeile>[,[<Spalte>] [,[<Cursor>] [, [<Anfang>] [,<Ende>]]]]

Verwendung: Positioniert den Cursor auf dem Bildschirm, definiert seine Größe und macht ihn sichtbar bzw. unsichtbar.

Bemerkungen:

<Zeile>
Gibt die Zeilennummer auf dem Bildschirm in der der Cursor angezeigt werden soll. Die Nummer liegt in dem Bereich von 1 bis 25.

<Spalte>
Gibt die Spaltennummer auf dem Bildschirm. Diese Nummer liegt in einem Bereich von 1 bis 40 (bei einer Bildschirmbreite von 40) oder von 1 bis 80 (bei einer Bildschirmbreite von 80).

<Cursor>
Ein Wert, mit dem angegeben wird, ob der Cursor sichtbar sein soll oder nicht. Soll er sichtbar sein, so wird 1, ansonsten 0 angegeben. <Cursor> gilt für den Grafik-Modus nicht.

<Anfang>
Eine Zeichenposition auf dem Bildschirm besteht aus 14 oder 8 Suchzeilen, je nachdem, ob ein Adapter mit einfarbigem Bildschirm oder mit Farbbildschirm benutzt wird. Die Suchzeilen werden ab dem Anfang der Zeichenposition von 0 bis 13 bzw. 0 bis 7 numeriert. Mit <Anfang> wird die oberste Suchzeile des Cursors festgelegt.

<Ende>
Legt die unterste Zeile für den Cursor fest. Der Bereich zwischen "Anfang" und "Ende" wird an der Zeichenposition aufgefüllt, an der der Cursor steht. Wird für "Anfang" eine höhere Suchzeilennummer als für "Ende" angegeben, so springt der Cursor zu dem oberen Teil der Zeichenposition, so daß ein geteilter Cursor angezeigt wird. "Anfang"
und <Ende> werden im Grafik-Modus nicht benutzt.

Durch die Cursor-Position wird die Stelle auf dem Bildschirm festgelegt, an der das Zeichen angezeigt wird.

Jeder der Befehls-Parameter kann weggelassen werden. Für einen weggelassenen Parameter gilt weiter der aktuelle Wert.

Wird die Anzeige der Funktionstasten in der untersten Bildschirmzeile ausgeschaltet (KEY OFF), so können alle 25 Zeilen benutzt werden. GW-BASIC schreibt normalerweise nicht in Zeile 25. Bewegen Sie den Cursor jedoch in diese Zeile, so schreibt es auch dort.

Bei der Ausführung eines Programms schaltet GW-BASIC den Cursor normalerweise aus. Mit dem Befehl LOCATE „,1 wird er wieder eingeschaltet.

Beispiele:

LOCATE 1,1

Bewegt den Cursor zu der oberen linken Ecke des Anzeigenbereichs für Zeichen.

LOCATE „,0,7

Legt einen Block-Cursor auf einem Farbbildschirm fest, ändert jedoch die Position oder Anzeige des Cursors nicht.

LOCATE 24,1,1,6,1

Setzt einen geteilten sichtbaren Cursor an den Anfang der 24sten Bildschirmzeile.

Hinweis:

Zulässige Bereiche für die LOCATE-Parameter sind 1 bis 25 für <Zeile>, 1 bis maximale Bildschirmbreite für <Spalte>, 0 oder 1 für <Cursor> und 1 bis 31 für <Anfang> bzw. <Ende>. Werden außerhalb der zulässigen Bereiche liegende Werte angegeben, so wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt.

Es kommt gelegentlich vor, daß der Cursor nach einem Befehl, der den Anzeige-Modus oder den

Cursor selbst beeinflußt, nicht wieder erscheint. In diesem Fall genügt

LOCATE „1

Somit wird der Cursor wieder sichtbar.

LOF-Funktion

Syntax: LOF(<Dateinummer>)

Verwendung:
Gibt die Länge der Datei in Bytes zurück.

Bemerkungen: LOF kann auch für eine Datenübertragungsdatei benutzt werden. In diesem Fall gibt die Funktion den noch in dem Eingabepuffer freien Platz zurück. Der Höchstwert des freien Platzes beträgt normalerweise 256 Bytes. Dieser Standardwert kann jedoch beim Laden von GW-BASIC mit der Option /C geändert werden.

Beispiel: Das folgende Beispiel liest den letzten Satz einer Direktzugriffsdatei in den Puffer. Die Satzlänge muß schon in RECSIZ% gespeichert worden sein:

```
10 OPEN "AFILE" AS #1
20 GET #1,LOF(1)/RECSIZ%
```

Hinweis: Wird LOF für Dateien benutzt, die unter IBM BASIC 1.10 erstellt wurden, so ist die zurückgegebene Länge ein Vielfaches von 128. Eine echte Länge von 290 ergibt beispielsweise ein Ergebnis von 384.

LOG-Funktion

Syntax: LOG(X)

Verwendung: Gibt den natürlichen Logarithmus von X zurück. X muß größer sein als Null.

Beispiel: PRINT LOG(45/7)
ergibt
1.860752

Hinweis: LOG wird mit einfacher Genauigkeit ausgeführt, es sei denn, Sie geben beim Laden von GW-BASIC die Option /D an. Bei dieser Option werden LOG und andere "residente" Funktionen mit doppelter Genauigkeit berechnet.

LPOS-Funktion

Syntax: LPOS(X)

wobei X das Kennzeichen für den Zeilendrucker darstellt.

Verwendung: Gibt das aktuelle Zeichen in dem Druckpuffer zurück, das für das Ausdrucken bereit steht.

Bemerkungen: LPOS gibt nicht unbedingt die physikalische Position des Druckkopfes an.

Mit dem Wert von X wird bestimmt, welcher Drukker getestet wird:

0 oder 1	LPT1:
2	LPT2:
3	LPT3:

Beispiel: 100 IF LPOX(X)>60 THEN LPRINT CRH$(13)

Diese Programmzeile gewährleistet, daß nicht mehr als 60 Zeichen auf einer Zeile gedruckt werden. CHR$(13) erzeugt eine Zeilenschaltung <CR>.

LPRINT- und LPRINT USING-Befehle

Syntax: LPRINT [<Liste mit Ausdrücken>][;]

LPRINT USING <Zeichenfolgenausdruck>; <Liste mit Ausdrücken> [;]

Verwendung: Druckt Daten auf dem Zeilendrucker (LPT1:) aus.

Bemerkungen: <Liste mit Ausdrücken>
enthält die Elemente, die ausgedruckt werden müssen. Bei diesen Elementen handelt es sich um numerische Ausdrücke und/oder Zeichenfolgenausdrücke. Sie müssen durch Kommas oder Semikolon voneinander getrennt werden.

<Zeichenfolgenausdruck>
ist eine Zeichenfolgenkonstante oder Variable, mit der das für das Ausdrucken benutzte Format angegeben wird.

Die einzelnen Angabe bezüglich der Druckformate entsprechen den Angaben für die Anzeige von Zeichen auf dem Bildschirm. Hier wird auf PRINT und PRINT USING verwiesen.

Bei LPRINT wird davon ausgegangen, daß ein Drucker mit einer Zeilenbreite von 80 Zeichen benutzt wird. Die Folge für Zeilenschaltung/Zeilenvorschub wird dementsprechend automatisch eingefügt. Die Zeilenbreite kann mit dem Befehl WIDTH"LPT1:" geändert werden.

LPRINT gibt eine Fehlermeldung "Device Timeout" (Zeitsperre bei der Einheit) aus, wenn der Drucker oder eine andere Einheit, die die LPRINT-Ausgabe empfängt, mit der Antwort in Verzug ist. Sie können der Einheit mehr Zeit geben, indem Sie eine Unterbrechung angeben. Zum Beispiel:

```
9900 IF ERR = 24 THEN RESUME
```

(Fehler-Nr. 24 ist die Nummer für die Fehlermeldung "Device Timeout" (Zeitsperre bei der Einheit).

Hinweis:

Die Benutzung von LPRINT ist nicht auf druckbare Zeichen beschränkt. Das Programm kann LPRINT zur Aktivierung von Druckerfunktionen benützen, wie beispielsweise zur Bewegung des Druckkopfes und Auswahl der Schriftart. Hierzu gibt es eine Reihe von Standardcodes, z.B.:

```
LPRINT CHR$(12);
```

Erzeugt einen Seitenvorschub bei dem Drucker. Weitere Codes werden an den benutzten Drucker weitergegeben. Für diese Codes wird auf die Begleitdokumentation zu dem Drucker verwiesen.

LSET- und RSET-Befehle

Syntax:

LSET <Zeichenfolgenvariable>=
<Zeichenfolgenausdruck>
RSET <Zeichenfolgenvariable>=
<Zeichenfolgenausdruck>

Verwendung:

Überträgt Daten aus dem Speicher in einen Pufferspeicher für Direktzugriffsdateien (Vorbereitung für einen PUT-Befehl).

Bemerkungen:

<Zeichenfolgenvariable>
ist eine Variable, die schon von FIELD definiert wurde.

<Zeichenfolgenausdruck>
enthält die Informationen, die bei der durch die Zeichenfolgenvariable angegebenen Position in den Pufferspeicher für die Direktzugriffsdateien gesetzt werden müssen.

Erfordert <Zeichenfolgenausdruck> weniger Bytes als für <Zeichenfolgenvariable> angegeben, so richtet LSET die Zeichenfolge linksbündig in dem Feld aus, während RSET sie rechtsbündig ausrichtet. (Zusätzliche Positionen werden mit Leerzeichen aufgefüllt.) Ist die Zeichenfolge zu lang für das Feld, so werden Zeichen am rechten Ende abgeschnitten. Numerische Werte müssen in Zeichenfolgen umgewandelt werden, bevor sie mit LSET oder RSET übertragen werden (siehe MKI$, MKS$, MKD$).

Beispiele:

```
150 LSET W$ = "Very + WEATHER$
```

Setzt den Zeichenfolgenausdruck bei der durch die FIELD-Variable W$ angegebenen Position in den Puffer. Die Zeichenfolge wird linksbündig in dem FIELD-Bereich ausgerichtet. Gegebenenfalls wird sie auf der rechten Seite dieses Bereich mit Leerzeichen aufgefüllt oder abgeschnitten.

In dem folgenden Beispiel wird ein numerischer Wert in eine Zeichenfolge umgewandelt, bevor er rechtsbündig ausgerichtet in den Puffer gesetzt wird:

```
30 ABC$ = MKI$(14)
40 RSETA$ = ABC$
```

Hinweis:

LSET und RSET können auch für nicht mit FIELD angegebene Variablen benutzt werden. Dies ist beim Formatieren von Texten besonders nützlich, die auf dem Bildschirm angezeigt oder ausgedruckt werden sollen.

MERGE-Befehl

Syntax: MERGE<Dateispez>

Verwendung: Mischt eine angegebene Diskettendatei mit dem gerade im Speicher stehenden Programm.

Bemerkungen: Die <Dateispez> muß den Dateinamen enthalten, der beim Sichern der Datei benutzt wurde. Die Datei muß im ASCII-Format gesichert worden sein (siehe SAVE). Ist dies nicht der Fall, so wird eine Fehlermeldung "Bad file mode" (Falscher Datei-Modus) angezeigt. Die "Dateispez" kann einen Pfadnamen enthalten.

Verfügen Zeilen in der Diskettendatei über dieselben Zeilennummern wie Zeilen in dem Programm im Speicher, so ersetzen die Zeilen der Datei auf Diskette die entsprechenden Zeilen im Speicher.

Nach Ausführung eines MERGE-Befehls kehrt GW-BASIC stets auf Befehlsebene ("Ok") zurück.

Beispiel: MERGE "NUMBERS"

Mischt die GW-BASIC-Programmdatei NUMBERS.BAS in dem gegenwärtigen Inhaltsverzeichnis der aktuellen Diskette mit dem schon im Speicher stehenden Programm.

MID$-Befehl

Syntax: MID$(<Zeichenfolgenausdruck1>,n[,m])= <Zeichenfolgenausdruck2>

wobei n und m ganzzahlige Ausdrücke darstellen.

Verwendung: Ersetzt einen Teil einer Zeichenfolge durch eine andere Zeichenfolge.

Bemerkungen: Die Zeichen in <Zeichenfolgenausdruck1> werden ab Position n durch die Zeichen in "Zeichenfolgenausdruck2" ersetzt. Das wahlweise "m" bezieht sich auf die Anzahl von Zeichen in "Zeichenfolgenausdruck2", die als Ersatz benutzt werden. Wird "m" weggelassen, so wird der gesamte <Zeichenfolgenausdruck2> benutzt. Unabhängig davon, ob "m" angegeben wird oder nicht, der Austausch von Zeichen geht niemals über die Länge von <Zeichenfolgenausdruck1> hinaus.

Beispiel: Siehe MID$-Funktion.

Hinweis: Die Werte n und m müssen in dem Bereich von 1 bis 255 liegen. Ansonsten wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt.

MID$-Funktion

Syntax: MID$(X$,n[,m])

Verwendung: Gibt eine Zeichenfolge mit einer Länge von m Zeichen aus X$ zurück, wobei mit dem n-ten Zeichen begonnen wird.

Bemerkungen: n und m müssen ganzzahlige Ausdrücke in dem Bereich von 1 bis 255 sein. Wird m weggelassen oder sind weniger als m Zeichen rechts von dem n-ten Zeichen vorhanden, so werden sämtliche rechtsbündigen Zeichen ab dem n-ten Zeichen zurückgegeben. Ist n größer als die Anzahl von Zeichen in X$ (LEN(X$)), so gibt MID$ eine Nullfolge zurück.

Beispiel:

```
10 A$ = "GUTEN"
20 B$ = "MORGEN MITTAG ABEND"
30 PRINT A$;MID$(B$,9,7)
```

ergibt

```
GUTEN ABEND
```

Die MID$-Funktion wird insbesondere dazu benutzt, die Zeichen einer Zeichenfolge nacheinander zu überprüfen. Mit dem folgenden Programm werden Sie aufgefordert, einen Text einzugeben. Das Programm prüft, ob in dieser Zeichenfolge Zeichen vorhanden sind, bei denen es sich nicht um Buchstaben handelt. Sobald ein derartiges Zeichen gefunden wird, wird seine Position auf dem Bildschirm angezeigt.

```
10 INPUT "Bitte geben Sie einen Text ein";T$
20 IF LEN(T$) = 0 THEN GOTO 10
30 FOR L% = 1 TO LEN(T$)
40 CHAR$ = MID$(T$,L%,1)
50 IF CHAR$<"A" OR CHAR$>"z" OR
   CHAR$>"Z" AND CHAR$<"a" THEN
   P% = L%: GOTO 100
60 NEXT L%
70 PRINT "Der Text besteht nur aus
   Buchstaben":END
```

```
100 PRINT "Ein anderes Zeichen als ein
    Buchstabe"; CHAR$; "wurde in Position
    gefunden"; P%
110 END
```

Hinweis: Liegt n oder m nicht in dem zulässigen Bereich, so wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt.

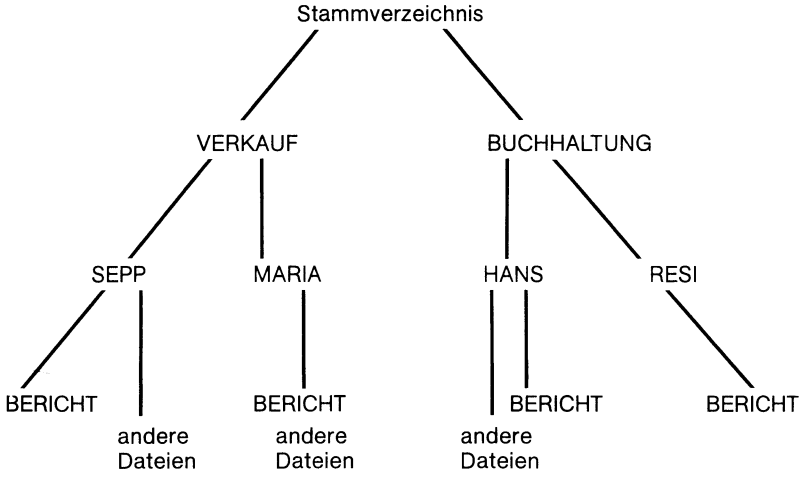
MKDIR-Befehl

Syntax: <Pfad>

Verwendung: Erstellt ein Inhaltsverzeichnis in dem angegebenen Laufwerk.

Bemerkungen: <Pfad> ist ein Zeichenfolgenausdruck mit nicht mehr als 128 Zeichen, der ein neues, zu erstellendes Inhaltsverzeichnis angibt. Für Einzelheiten über Pfade und Inhaltsverzeichnisse wird auf das NCR-DOS Handbuch verwiesen.

Beispiele: Wird von dieser hierarchischen Struktur ausgegangen und befinden Sie sich gerade in dem Stammverzeichnis, so erstellt der Befehl:



MKDIR "FORSCHUNG"

ein Unterverzeichnis mit diesem Namen in dem Stammverzeichnis (d.h. auf derselben Ebene wie VERKÄUFE und BUCHHALTUNG).

MKDIR "FORSCHUNG\MARTHA"

erstellt ein Unterverzeichnis namens MARTHA in dem Inhaltsverzeichnis FORSCHUNG.

Hinweis: MKDIR ist identisch mit dem entsprechenden NCR-DOS Befehl.

MKI$-, MKS$-, MKD$-Funktionen

Syntax:

MKI$ (<ganzzahliger Ausdruck>)
MKS$ (<Ausdruck mit einfacher Genauigkeit>)
MKD$ (<Ausdruck mit doppelter Genauigkeit>)

Verwendung:

Wandelt numerische Werte in Zeichenfolgenwerte um.

Bemerkungen:

Jeder numerische Wert, der mit einem LSET- oder RSET-Befehl in den Puffer für Direktzugriffsdateien gesetzt werden soll, muß zuerst in eine Zeichenfolge umgewandelt werden. Mit MKI$ wird eine Ganzzahl in eine aus 2 Bytes bestehende Zeichenfolge umgewandelt. Mit MKS$ wird eine Zahl mit einfacher Genauigkeit in eine Zeichenfolge mit 4 Bytes umgewandelt. Mit MKD$ wird eine Zahl mit doppelter Genauigkeit in eine aus 8 Bytes bestehende Zeichenfolge umgewandelt.

Beispiel:

```
90 AMT=(K+T)
100 FIELD #1,4 AS D$20 AS N$
110 LSET D$=MKS$(AMT)
120 LSET N$=A$
130 PUT #1
```

Wandelt den Wert mit einfacher Genauigkeit in AMT in eine aus 4 Bytes bestehende Zeichenfolge um, setzt diese Zeichenfolge in den Puffer für Direktzugriffsdateien und schreibt sie als Teil des Datensatzes in eine Datei.

Hinweis:

Die entgegengesetzten Funktionen d.h. die Funktionen, mit denen Zeichenfolgen in numerische Werte umgewandelt werden, sind CVI, CVS und CVD.

Die STR$-Funktion wandelt einen numerischen Wert ebenfalls in einen Zeichenfolgenwert um, behält jedoch nicht unbedingt die Länge der Originalzahl in Bytes bei.

NAME-Befehl

Syntax: NAME <alter Dateiname> AS <neuer Dateiname>

Verwendung: Ändert den Namen einer Diskettendatei.

Bemerkungen: <alter Dateiname> muß vorhanden sein, während <neuer Dateiname> nicht vorhanden sein darf; ansonsten kommt es zu einem Fehler.

Eine Datei darf nicht mit einer neuen Laufwerkbezeichnung umbenannt werden. Wird dies versucht, so wird eine Fehlermeldung "Rename across disk" (Umbenennung bei verschiedenen Disketten) angezeigt. Nach einem NAME-Befehl steht die Datei auf derselben Diskette in demselben Diskettenbereich, hat jedoch einen neuen Namen.

Beispiel: NAME "BUCHH" AS "HAUBTB"

In diesem Beispiel wird die früher mit BUCHH benannte Datei in HAUPTB umbenannt.

Hinweis: Bei dem NAME-Befehl geht GW-BASIC nicht von der Dateinamenerweiterung .BAS aus.

Die Angabe eines Pfadnamens ist nicht zulässig.

Bevor NAME benutzt wird, muß die Datei abgeschlossen werden.

NEW-Befehl

Syntax: NEW

Verwendung: Löscht das gerade im Speicher stehende Programm, schließt sämtliche Dateien ab und löscht alle Variablen. Ist TR (Ablaufverfolgung) ON, so wird diese Funktion ausgeschaltet.

Bemerkungen: In NEW wird im Direkt-Modus gegangen, um den Speicher zu löschen, bevor ein neues Programm eingegeben wird. Nach Ausführung eines NEW-Befehls geht GW-BASIC stets auf Befehlsebene zurück.

Durch NEW werden sämtliche Dateien abgeschlossen. Die Ablaufverfolgung wird ausgeschaltet.

OCT$-Funktion

Syntax: OCT$(X)

Verwendung: Gibt eine Zeichenfolge zurück, die den Oktalwert des Dezimalparameters darstellt. X wird auf eine Ganzzahl abgerundet, bevor OCT$(X) ausgewertet wird.

Beispiel:
```
PRINT OCT$(24)
ergibt
 30
```

Für Einzelheiten über die Hexadezimal-Umwandlung wird auf die HEX$-Funktion verwiesen.

ON COM(n)-Befehl

Syntax: ON COM(n) GOSUB <Zeile>

n
Nummer des Datenübertragungskanals (1 oder 2).

Zeile
Zeilennummer für den Beginn der Verzweigungsroutine. Durch eine Zeilennummer 0 wird die Verzweigung für den angegebenen Kanal deaktiviert.

Funktion: Mit diesem Befehl kann eine Verzweigung für die Datenübertragung festgelegt werden. Sobald Daten in den Datenübertragungspuffer gesetzt werden, verzweigt sich GW-BASIC bei der angegebenen Zeilennummer zu der Subroutine.

Bemerkungen: Die folgenden Befehle steuern die Aktivierung oder Deaktivierung der Verzweigung bei der Datenübertragung:

COM(n) ON
Muß ausgeführt werden, um den ON COM(n) Befehl zu aktivieren. Wird eine von Null abweichende Zeilennummer in ON COM(n) angegeben, so prüft GW-BASIC bei jedem Starten eines neuen Befehls durch das Programm, ob Zeichen in den angegebenen Kanal gesetzt wurden. Ist mindestens ein Zeichen vorhanden, so führt GW-BASIC einen GOSUB-Befehl zu der angegebenen Zeile aus.

COM(n) OFF
Wird dieser Befehl ausgeführt, so erfolgt keine Programmverzweigung bei dem Kanal. Kommen Daten in dem Übertragungskanal an, so werden sie von GW-BASIC nicht festgehalten.

COM(n) STOP
Wird dieser Befehl ausgeführt, so findet bei dem Kanal keine Programmverzweigung statt. Die in dem Kanal empfangenen Zeichen werden jedoch im Speicher gesichert, damit bei Ausführung von COM(n) ON sofort eine Programmverzweigung stattfindet.

Findet eine Verzweigung statt, so führt GW-BASIC automatisch einen COM(n) STOP Befehl aus, damit es nicht zu sich wiederholenden Programmverzweigungen kommen kann. Beim Rücksprung aus der Unterbrechungsroutine wird automatisch COM(n) ON ausgeführt, es sei denn, innerhalb der Unterbrechungsroutine wurde explizit COM(n) OFF ausgeführt.

Eine Programmverzweigung findet nur statt, wenn GW-BASIC ein Programm ausführt.

Kommt es zu einer Unterbrechung wegen eines Fehlers (siehe ON ERROR), so werden sämtliche Verzweigungen automatisch deaktiviert. Dies bedeutet, daß Übertragungsaktivitäten von GW-BASIC nicht berücksichtigt werden.

Typischerweise liest die Unterbrechungsroutine bei der Datenübertragung vor dem Rücksprung eine ganze Meldung aus dem Datenübertragungskanal. Es wird nicht empfohlen, eine Programmverzweigung bei aus einem Zeichen bestehenden Meldungen zu benutzen, da der Aufwand für die Unterbrechung und das Lesen einzelner Zeichen bei hohen Baudraten zu einem Überlauf des Unterbrechungspuffers für die Datenübertragung führen kann.

Mit dem RETURN-Befehl am Ende der Unterbrechungsroutine kann wahlweise eine Zeilennummer angegeben werden. Mit ihr wird bei der angegebenen Zeilennummer in das Programm zurückgegangen. Dadurch wird der GOSUB-Eintrag eliminiert, den die Programmverweigung erstellt hat. RETURN mit Angabe einer Zeilennummer sollte mit Vorsicht benutzt werden! Jeder andere GOSUB-, WHILE- oder FOR-Befehl, der zum Zeitpunkt der Unterbrechung aktiv war, bleibt aktiv.

Beispiel:

```
30 ON COM(1) GOSUB 9900
40 COM(1) ON
 .
 .
 .
```

```
9900 REM Unterbrechungsroutine für
     ankommende Zeichen
.
.
.
9990 RETURN
```

Wurde die Übertragungsaktivität und Programmverzweigung für diesen Datenübertragungskanal zum Zeitpunkt des Ereignisses aktiviert, so bedeutet Zeile 30, daß sich GW-BASIC bei Zeile 9900 zu der Subroutine verzweigt. Mit Zeile 40 wird die Verzweigung für diesen Kanal aktiviert.

ON ERROR GOTO-Befehl

Syntax: ON ERROR GOTO <Zeilennummer>

Verwendung: Aktiviert die Fehlerbehandlung und gibt die erste Zeile der Fehlerbehandlungsroutine an.

Bemerkungen: Nachdem die Fehlerbehandlung aktiviert wurde, wird bei sämtlichen entdeckten Fehlern, einschließlich der Fehler im Direkt-Modus, zu der angegebenen Fehlerbehandlungsroutine gesprungen. Wird keine <Zeilennummer> angegeben, so kommt es zu der Fehlermeldung "Undefined line" (Undefinierte Zeile).

Zur Deaktivierung der Fehlerbehandlung wird ON ERROR GOTO 0 ausgeführt. Bei nachfolgenden Fehlern wird eine Fehlermeldung ausgedruckt und die Ausführung angehalten. Bei einem ON ERROR GOTO 0-Befehl in einer Fehlerbehandlungsroutine stoppt GW-BASIC und druckt die Fehlermeldung für den Fehler aus, der die Unterbrechung verursacht hat. Es wird empfohlen, daß sämtliche Fehlerbehandlungsroutinen ON ERROR GOTO 0 ausführen, wenn ein Fehler angetroffen wird, für den keine fehlerbehenbende Maßnahme vorhanden ist.

Kommt es während der Ausführung einer Fehlerbehandlungsroutine zu einem Fehler, so wird die entsprechende Fehlermeldung ausgedruckt und die Ausführung beendet. Innerhalb der Fehlerbehandlungsroutine kommt es zu keiner Unterbrechung wegen eines Fehlers.

Beispiele:

```
10 ON ERROR GOTO 500
 .
 .
 .
500 REM Fehlerbehandlungsroutinen
510 ON ERROR GOTO 0
520 IF ERR = 24 THEN RESUME 'Einheit zu
    langsam
 .
 .
 .
```

```
690 RESUME
```

Mit Zeile 10 wird angegeben, daß sich GW-BASIC bei Fehlern zu Zeile 500 verzweigen soll. Mit Zeile 520 wird angegeben, daß GW-BASIC wieder zurückgehen soll, um erneut einen Verbindungsaufbau zu versuchen, wenn eine Datenübertragungs-Einheit nicht rechtzeitig geantwortet hat. Dies ist nur ein Beispiel für einen Fehler. In Anhang C werden noch verschiedene Fehlermöglichkeiten beschrieben, die bei einem zu erstellenden Programm berücksichtigt werden können.

Möglicherweise möchten Sie durch ein akustisches Signal auf einen Fehler aufmerksam gemacht werden. In diesem Fall gehen Sie folgendermaßen vor:

```
10 ON ERROR GOTO 9900
 .
 .
 .
9900 REM Akustisches Signal, bis eine Taste
     betätigt wird
9910 BEEP
9920 IF INKEY$ = ""THEN 9910
9930 END
```

ON...GOSUB- und ON...GOTO-Befehle

Syntax: ON <Ausdruck> GOTO <Liste mit Zeilennummern>

ON <Ausdruck> GOSUB <Liste mit Zeilennummern>

Verwendung: Verzweigung zu einer von mehreren angegebenen Zeilennummern, je nach dem bei Auswertung von <Ausdruck> zurückgegebenen Wert.

Bemerkungen: Der Wert von <Ausdruck> bestimmt, welche Zeilennummer in der Liste für die Verzweigung benutzt wird. Ist der Wert beispielsweise 3, so wird die Verzweigung zu der dritten Zeilennummer in der Liste vorgenommen. (Handelt es sich bei dem Wert nicht um eine Ganzzahl, so wird der Bruchteil abgerundet.)

In dem ON...GOSUB-Befehl muß jede Zeilennummer in der Liste die erste Zeilennummer einer Subroutine darstellen. Wird ON...GOSUB benutzt, so muß gewährleistet sein, daß GW-BASIC nach Ausführung der Subroutine wieder in das Hauptprogramm zurückfindet. Dies erfolgt über den RETURN-Befehl.

Ist der Wert von "Ausdruck" gleich Null oder größer als die Anzahl von Elementen in der Liste (jedoch nicht größer als 255), so führt GW-BASIC mit dem nächsten ausführbaren Befehl fort. Ist der Wert des Ausdrucks negativ oder größer als 255, so kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

Beispiele: 100 ON L-1 GOTO 150,300,320,390

Ist der Wert des Ausdrucks nach einer erforderlichen Abrundung gleich 1, so verzweigt sich GW-BASIC zu Zeile 150. Ist dieser Wert gleich 2, so verzweigt sich GW-BASIC zu Zeile 300 usw. Ist der Wert von L-1 größer als 4, so geht GW-BASIC einfach zu der nächsten Zeile nach 100 weiter.

Das folgende Programm fordert Sie auf, je nach der

auszuführenden einfachen arithmetischen Operation eine Zahl von 1 bis 4 einzugeben. Mit Zeile 110 wird GW-BASIC aufgefordert, je nach der von Ihnen getroffenen Wahl in eine von vier Subroutinen zu gehen. Nachdem es die Rechenaufgabe ausgeführt und Ihnen das Ergebnis mitgeteilt hat, kehrt GW-BASIC mit RETURN zu dem Befehl zurück, der auf den ON...GOSUB Befehl folgt. Betätigen Sie eine beliebige Taste, um zu dem Anfang des Programms zurückzugehen. Wird eine Zahl 0 eingegeben, so wird das Programm beendet.

```
10 CLS:PRINT "Lassen Sie mich Ihre Rechen-
   aufgaben erledigen":PRINT
20 INPUT "Geben Sie die erste von zwei Zahlen
   ein";N1
30 INPUT "Nun geben Sie die zweite Zahl ein":N2
40 IF N1 = 0 OR N2 = 0 THEN END
50 CLS:PRINT "Betätigen Sie 1,2,3 oder
   4":PRINT
60 PRINT "1) Addieren Sie die Zahlen"
70 PRINT "2) Subtrahieren Sie die zweite Zahl
   von der ersten"
80 PRINT "3) Multiplizieren Sie die Zahlen"
90 PRINT "4) Dividieren Sie die erste Zahl
   durch die zweite"
100 INPUT CHOICE:PRINT
110 ON CHOICE GOSUB 150,170,190,210
120 IF INKEY$=""THEN 120
130 GOTO 10
140 REM ****** die Subroutinen
150 PRINT N1;"plus";N2;"gleich";N1+N2
160 RETURN
170 PRINT N1;"minus";N2:"gleich";N1-N2
180 RETURN
190 PRINT N1;"mal";N2;"gleich";N1*N2
200 RETURN
210 PRINT N1;"dividiert durch";N2;"gleich";
    N1/N2
220 RETURN
```

ON KEY-Befehl

Syntax: KEY(n) GOSUB <Zeilennummer>

n ist eine Zahl in dem Bereich von 1 bis 20, die sich auf eine programmierbare Funktionstaste, eine Cursor-Taste oder eine vom Benutzer definierte Tastenkombination bezieht, bei der es zu einer Programmverzweigung kommt:

1-10	Die zehn Funktionstasten
11	Cursor nach oben
12	Cursor nach links
13	Cursor nach rechts
14	Cursor nach unten
15-20	Vom Benutzer definierte Tastenkombination (siehe KEY).

<Zeilennummer> entspricht der ersten Zeile der Subroutine, in die sich GW-BASIC verzweigt, wenn die mit n angegebene Taste betätigt wird. Ist <Zeilennummer> gleich 0, so führt die Betätigung dieser Taste nicht zu einer Verzweigung.

Verwendung: Mit diesem Befehl wird GW-BASIC angewiesen, sich bei einer bestimmten Zeile in eine Subroutine zu verzweigen, wenn eine bestimmte Taste oder Tastenkombination betätigt wird.

Bemerkungen: Die folgenden Befehle steuern die Aktivierung oder Deaktivierung der Programmverzweigung bei einer Tastenbetätigung.

KEY(n) ON
Muß ausgeführt werden, wenn ON KEY(n) eine Auswirkung haben soll. Wird für die Programmverzweigung mit ON KEY(n) eine von Null abweichende Zeilennummer angegeben, so prüft GW-BASIC bei jedem Starten eines neuen Befehls, ob die angegebene Taste betätigt wurde. Wurde die Taste betätigt, so führt GW-BASIC einen GOSUB-Befehl zu der angegebenen Zeile aus.

KEY(n) OFF
Wird dieser Befehl ausgeführt, so findet bei Betäti-

gung der angegebenen Taste keine Verzweigung statt. Wird die Taste betätigt, so reagiert GW-BASIC nicht.

KEY(n) STOP
Wird dieser Befehl ausgeführt, so findet keine Verzweigung statt. Wird die angegebene Taste jedoch betätigt, so findet bei Ausführung von KEY(n) ON eine sofortige Verzweigung statt.

Kommt es zu einer Verzweigung, so veranlaßt GW-BASIC automatisch einen KEY(n) STOP Befehl für die unterbrochene Taste, damit es keinesfalls zu sich wiederholenden Verzweigungen kommen kann. Beim Rücksprung aus der Unterbrechungsroutine (mit RETURN) wird automatisch ein KEY(n) ON Befehl ausgeführt, es sei denn, innerhalb der Unterbrechungsroutine wurde ein expliziter KEY(n) OFF Befehl ausgeführt.

Eine Verzweigung findet nur statt, wenn GW-BASIC ein Programm ausführt.

Kommt es zu einer Unterbrechung wegen eines Fehlers, so werden sämtliche Verzweigungen automatisch deaktiviert. Dies bedeutet, daß Tastatur-Ereignisse ignoriert werden.

Befindet sich GW-BASIC im Direkt-Modus, so werden keine Programmverzweigungen aktiviert. Insbesondere die Funktionstasten führen ihre programmierte Funktion aus.

Eine Taste, die zu einer Programmverzweigung führt, kann später nicht mit INPUT oder INKEY$ getestet werden, so daß für jede Taste eine andere Unterbrechungsroutine benutzt werden muß, wenn verschiedene Funktionen gewünscht werden.

Mit dem RETURN-Befehl am Ende der Unterbrechungsroutine kann wahlweise eine Zeilennummer angegeben werden. Mit RETURN "Zeile" wird bei einer festgelegten Zeilennummer in das Programm zurückgegangen. Durch diese Aktion wird der GOSUB-Eintrag eliminiert, den die Verzweigung erstellt hat. RETURN <Zeile> muß mit Vorsicht benutzt werden! Jeder andere GOSUB-, WHILE-

oder FOR-Befehl, der zum Zeitpunkt der Verzweigung aktiv war, bleibt aktiv.

Beispiel:

Durch das folgende Beispiel wird die normale Auswirkung von <Ctrl-Break> verhindert:

```
10 KEY 20,CHR$(&H04)+CHR$(70)
20 ON KEY(20) GOSUB 1000
30 KEY(20) ON
40 PRINT "Versuchen Sie, zu "Ok" zurückzukehren
50 GOTO 40
1000 PRINT "Sie haben erfolglos versucht, auszubrechen"
1010 RETURN
```

Bevor dieses Programm ausgeführt wird, muß geprüft werden, ob andere im Speicher stehende Programme auf Diskette gesichert wurden, da für das Ausbrechen ein Neustart des Systems mit <Ctrl-Alt-Del> erforderlich ist.

ON PEN-Befehl

Syntax: ON PEN GOSUB <Zeilennummer>

Zweck: Dieser Befehl legt eine Zeilennummer fest, in die sich GW-BASIC bei Benutzung des Lichtstiftes verzweigt.

Bemerkungen: <Zeilennummer> ist die erste Zeile der Subroutine, in die sich GW-BASIC verzweigt, wenn eine Aktivität des Lichtstiftes entdeckt wird. Wird eine <Zeilennummer> 0 angegeben, so wird die Verzweigung bei Benutzung des Lichtstiftes deaktiviert.

Wurde eine andere Zeilennummer als 0 angegeben und wurde ein PEN ON Befehl ausgeführt, so prüft GW-BASIC vor jeder Ausführung eines neuen Befehls, ob der Lichtstift aktiviert wurde. Wenn ja, verzweigt sich GW-BASIC bei "Zeilennummer" zu der Subroutine.

PEN OFF bedeutet, daß keine Verzweigung bei Benutzung des Lichtstiftes erfolgt. Darüber hinaus wird eine Aktivität des Lichtstiftes nicht festgehalten.

PEN STOP bedeutet, daß keine Verzweigung bei Benutzung des Lichtstiftes erfolgt, eine Aktivität des Lichtstiftes jedoch von GW-BASIC festgehalten wird. Deshalb kommt es bei Ausführung von PEN ON sofort zu einer Programmverzweigung, wenn der Lichtstift in der Zwischenzeit benutzt wurde.

Kommt es zu der Programmverzweigung, so führt GW-BASIC automatisch einen PEN STOP Befehl aus, damit es nicht zu sich wiederholenden Programmverzweigungen kommen kann. Beim Rücksprung aus der Unterbrechungsroutine (mit RETURN) wird PEN ON automatisch ausgeführt, es sei denn, die Subroutine enthält einen expliziten PEN OFF Befehl.

Eine Programmverzweigung bei bestimmten Er eignissen findet nur statt, wenn GW-BASIC ein Programm ausführt.

Kommt es zu einer Unterbrechung wegen eines Fehlers (siehe ON ERROR), so wird jede Programmverzweigung automatisch deaktiviert. Dies bedeutet, daß eine Benutzung des Lichtstiftes von GW-BASIC ignoriert wird.

Die PEN-Funktion ist nicht betroffen, wenn die Benutzung des Lichtstiftes zu einer Programmverzweigung führt.

Mit dem RETURN-Befehl am Ende der Unterbrechungsroutine kann wahlweise eine Zeilennummer angegeben werden. In diesem Fall geht GW-BASIC zu der angegebenen Zeilennummer zurück. Durch Angabe dieser Zeilennummerwird der GOSUB-Eintrag eliminiert, den die Programmverzweigung erstellt hat. Sie sollte jedoch mit Vorsicht benutzt werden! Jeder GOSUB-, WHILE- oder FOR-Befehl, der zum Zeitpunkt der Verzweigung aktiv war, bleibt aktiv.

Beispiel:

```
10 ON PEN GOSUB 1000
20 PEN ON
 .
 .
 .
1000 REM Verzweigung bei Benutzung des
     Lichtstiftes
 .
 .
 .
1190 RETURN
```

In diesem Beispiel stehen in den Zeilen 10 und 20 die Befehle, mit denen eine Programmverzweigung bei Benutzung des Lichtstiftes erstellt und aktiviert werden kann. Mit Zeile 1190 wird der Rücksprung (RETURN) aus der Unterbrechungsroutine bei Benutzung des Lichtstiftes angegeben.

ON PLAY-Befehl

Syntax: ON PLAY(n) GOSUB <Zeilennummer>

Verwendung: Aktiviert die Ausführung anderer GW-BASIC Befehle, während Hintergrundmusik gespielt wird.

Bemerkungen: n ist ein ganzzahliger Ausdruck in dem Bereich von 1 bis 32. Es kommt zu einer Programmverzweigung, wenn nur noch n Noten in dem Puffer für die Hintergrundmusik stehen.

<Zeilennummer> ist die erste Zeile der Subroutine, in die sich GW-BASIC verzweigt. Bei einer Zeilennummer 0 kommt es zu keiner Verzweigung.

Wird eine andere Zeilennummer als 0 angegeben und wurde ein PLAY ON Befehl ausgeführt, so prüft GW-BASIC vor Ausführung eines neuen Befehls, ob der Wert in dem Puffer für die Hintergrundmusik von n auf n-1 gegangen ist. Wenn ja, verzweigt sich GW-BASIC bei <Zeilennummer> zu der Subroutine.

Wird PLAY OFF angegeben, so findet keine Programmverzweigung bei der Hintergrundmusik mehr statt. Außerdem wird keine Aktivität der Hintergrundmusik festgehalten.

Bei PLAY STOP kommt es zu keiner Programmverzweigung bei der Hintergrundmusik mehr, jedoch wird die Aktivität der Hintergrundmusik von GW-BASIC festgehalten. Deshalb kommt es zu einer sofortigen Verzweigung, sobald PLAY ON ausgeführt wird, sofern in der Zwischenzeit eine Aktivität der Hintergrundmusik stattgefunden hat.

Kommt es zu einer Programmverzweigung, so führt GW-BASIC automatisch einen PLAY STOP Befehl aus, damit es nicht zu sich wiederholenden Programmverzweigungen kommen kann. Beim Rücksprung aus der Unterbrechungsroutine wird PLAY ON automatisch ausgeführt, es sei denn, die Subroutine enthält einen expliziten PLAY OFF Befehl.

Eine Unterbrechung bei bestimmten Ereignissen findet nur statt, wenn GW-BASIC ein Programm ausführt.

Kommt es zu einer Unterbrechung wegen eines Fehlers (siehe ON ERROR), so werden sämtliche Programmverzweigungen automatisch deaktiviert. Dies bedeutet, daß Musik-Aktivitäten von GW-BASIC ignoriert werden.

Mit dem RETURN-Befehl am Ende der Unterbrechungsroutine bei Musikaktivitäten kann wahlweise eine Zeilennummer angegeben werden. In diesem Fall geht GW-BASIC zu der angegebenen Zeilennummer zurück. Dadurch wird der GOSUB-Eintrag eliminiert, den die Programmverzweigung erstellt hat. Diese Zeilennummer sollte jedoch mit Vorsicht benutzt werden! Andere GOSUB-, WHILE- oder FOR-Befehle, die zum Zeitpunkt der Unterbrechung aktiv waren, bleiben aktiv.

Beispiel:

```
10 ON PLAY(5) GOSUB 1000
20 PLAY ON
.
.
.
1000 REM Folgendes während der Hintergrund-
     musik ausführen
.
.
.
1190 RETURN
```

Es kommt zu einer Programmverzweigung, wenn nur noch fünf Noten in dem Puffer für die Hintergrundmusik stehen.

Hinweis:

Zu einer Musik-Aktivität kann es nur kommen, wenn sich PLAY in dem Modus für die Hintergrundmusik und nicht in dem Modus für die Vordergrundmusik befindet. Eine Musik-Aktivität wird nicht ausgegeben, wenn der Puffer für die Hintergrundmusik leer ist.

Für n darf kein zu hoher Wert ausgewählt werden. So verursacht ON PLAY(32) beispielsweise soviele Programmverzweigungen, daß kaum noch Zeit für den Rest des Programms bleibt.

ON STRIG-Befehl

Syntax: ON STRIG(n) GOSUB <Zeilennummer>

wobei (n) eine Zahl darstellt, mit der einer von maximal vier Steuerknüppeln für Computerspiele angegeben werden kann. Gültige Zahlen sind die Zahlen 0, 2, 4 und 6.

<Zeilennummer> ist die Nummer der ersten Zeile einer Subroutine, die bei Betätigung des Steuerknüppels ausgeführt wird.

Verwendung: Gibt die erste Zeilennummer einer Subroutine an, die bei Betätigung eines Steuerknüppels ausgeführt wird.

Bemerkungen: Eine <Zeilennummer> 0 deaktiviert die Programmverzweigung bei einem Ereignis.

Der ON STRIG Befehl ist nur wirksam, wenn ein STRIG ON Befehl ausgeführt wurde (siehe STRIG-Befehl), um eine Programmverzweigung bei Benutzung dieses Steuerknüppels zu aktivieren. Ist die Programmverzweigung aktiviert und wird in ON STRIG eine andere Zeilennummer als 0 angegeben, so prüft GW-BASIC zwischen den Befehlen, ob der Steuerknüppel betätigt wurde. Wenn ja, wird ein GOSUB-Befehl zu der angegebenen Zeile ausgeführt.

Wurde ein STRIG OFF Befehl für den angegebenen Steuerknüppel ausgeführt (siehe STRIG-Befehl), so wird GOSUB nicht ausgeführt und GW-BASIC hält nicht fest, wenn der Steuerknüppel betätigt wurde.

Wurde ein STRIG STOP-Befehl für den angegebenen Steuerknüppel ausgeführt (siehe STRIG-Befehl), so wird der GOSUB-Befehl nicht ausgeführt. Sobald der entsprechende STRIG ON-Befehl ausgeführt wird, wird die Verzweigung ebenfalls ausgeführt.

Bei einer Programmverzweigung (d.h. GOSUB wird ausgeführt), wird automatisch ein STRIG STOP Befehl ausgeführt, damit es nicht zu sich wieder-

holenden Programmverzweigungen kommen kann. Beim Rücksprung aus der Unterbrechungsroutine wird automatisch ein STRIG ON-Befehl ausgeführt, es sei denn, innerhalb der Subroutine wurde ein expliziter STRIG OFF Befehl ausgeführt.

Mit dem RETURN-Befehl am Ende der Unterbrechungsroutine kann wahlweise eine Zeilennummer angegeben werden. In diesem Fall kehrt GW-BASIC zu der angegebenen Zeilennummer zurück. Diese Art des Rücksprungs muß jedoch mit Vorsicht benutzt werden, da andere GOSUB-, WHILE- oder FOR-Befehle, die zum Zeitpunkt der Programmverzweigung aktiv waren, aktiv bleiben.

Eine Programmverzweigung findet nur statt, wenn GW-BASIC ein Programm ausführt. Die Programmverzweigung wird automatisch deaktiviert, wenn es zu einer Unterbrechung wegen eines Fehlers kommt. In diesem Fall ignoriert GW-BASIC Aktivitäten des Steuerknüppels.

Die STRIG-Funktion ist von einer Programmverzweigung bei Benutzung des Steuerknüppels nicht betroffen.

Beispiel:

```
10 ON STRIG(2) GOSUB 2200
20 STRIG(2) ON
.
.
.
2200 REM befaßt sich mit der Betätigung eines
     Steuerknüppels
.
.
.
2290 RETURN
```

ON TIMER-Befehl

Syntax: ON TIMER(n) GOSUB <Zeilennummer>

Verwendung: Legt die Zeilennummer fest, bei der die Ausführung einer Subroutine nach Ablauf einer bestimmten Zeitspanne beginnt.

Bemerkungen: n ist ein numerischer Ausdruck in dem Bereich von 1 bis 86,400. Er stellt den Status des TIMER-Zählers dar, der eine Zeitgeber-Aktivität auslöst. Dieser Bereich von Sekunden bedeutet, daß eine Zeitspanne von bis zu 24 Stunden festgelegt werden kann. Im Anschluß an einen TIMER ON Befehl kommt es in Abständen von n Sekunden zu einer Zeitgeber-Aktivität.

<Zeilennummer> ist die erste Zeile der Subroutine, die GW-BASIC bei einer Zeitgeber-Aktivität ausführt. Wird eine Zeilennummer von 0 angegeben, so kommt es zu keiner Programmverzweigung bei Zeitgeber-Aktivitäten.

Nach Ablauf von n Sekunden kommt es zu der Zeitgeber-Aktivität. Danach zählt GW-BASIC wieder von 0 bis n und verzweigt sich bei der angegebenen Zeilennummer zu der Subroutine.

Wird TIMER OFF angegeben, so kommt es zu keiner Programmverzweigung bei Zeitgeber-Aktivitäten. Außerdem werden die Zeitgeber-Aktivitäten nicht festgehalten.

TIMER STOP bedeutet, daß es zu keiner Programmverzweigung bei Zeitgeber-Aktivitäten kommt, daß die Zeitgeber-Aktivität jedoch von GW-BASIC festgehalten wird. Aus diesem Grund kommt es bei Ausführung von TIMER ON sofort zu einer Programmverzweigung, wenn in der Zwischenzeit eine Zeitgeber-Aktivität stattfand.

Bei einer Programmverzweigung führt GW-BASIC automatisch einen TIMER STOP Befehl aus, damit es nicht zu sich wiederholenden Programmverzweigungen kommen kann. Beim Rücksprung aus der Subroutine wird automatisch TIMER ON

ausgeführt, es sei denn, die Subroutine enthält einen expliziten TIMER OFF Befehl.

Es kommt nur zu einer Programmverzweigung, wenn GW-BASIC ein Programm ausführt.

Kommt es zu einer Unterbrechung wegen eines Fehlers (siehe ON ERROR), so wird jede Programmverzweigung automatisch deaktiviert. Dies bedeutet, daß Zeitgeber-Aktivitäten von GW-BASIC nicht berücksichtigt werden.

Mit dem RETURN-Befehl am Ende der Subroutine kann wahlweise eine Zeilenummer angegeben werden. In diesem Fall geht GW-BASIC zu der angegebenen Zeilennummer zurück. Dadurch wird der GOSUB-Eintrag eliminiert, den die Programmverzweigung erstellt hat. Diese Art des Rücksprungs sollte jedoch mit Vorsicht benutzt werden! Andere GOSUB-, WHILE- oder FOR-Befehle, die zum Zeitpunkt der Programmverzweigung aktiv waren, bleiben aktiv.

Beispiel:

Das folgende Programm löst einmal pro Minute eine Zeitgeber-Aktivität aus. Der Bildschirminhalt wird in einer Matrixvariablen gesichert. Die laufende Zeit wird auf dem Bildschirm angezeigt. Danach wird der alte Bildschirminhalt wiederhergestellt. Die Zeilen 70 und 80 wurden aufgenommen, damit Sie Daten auf den Bildschirm schreiben können. (Sie können mit der Eingabe beginnen, sobald der Bildschirm im Anschluß an RUN gelöscht wird.) Gibt GW-BASIC ein akustisches Signal (Zeile 110) ab, so wird für einen Augenblick mit dem Schreiben eingehalten. Für die Speicherung des Bildschirminhalts in PIC$ (Zeilen 130 bis 160) wird etwas Zeit benötigt. Nehmen Sie den Schreibvorgang wieder auf, sobald der Bildschirm nach Anzeige der Zeit wiederhergestellt ist. Ggf. können Sie die Ihr die die Zeitanzeige in diesem Programm steuert, auf die gegenwärtige Zeit festlegen (siehe TIME$-Funktion).

```
10 DEFINT A-Z
20 DIM PIC$(24,80)
30 SCREEN 0:WIDTH 80
```

```
 40 ON TIMER(60) GOSUB 100
 50 TIMER ON
 60 CLS
 70 K$=INKEY$:IF K$="" THEN 70
 80 PRINT K$;
 90 GOTO 70
100 REM ***** Befaßt sich mit Zeitgeber-
    Aktivität
110 BEEP
120 ROW=CSRLIN:COL=POS(0)
130 FOR V=1 TO ROW
140 FOR H=1 TO 80
150 PIC$(V,H)=CHR$(SCREEN(V,H))
160 NEXT H:NEXT V
170 CLS:LOCATE 1,1
180 PRINT TIME$;
190 FOR DLY=1 TO 4000:NEXT DLY
200 CLS
210 FOR V=1 TO ROW
220 FOR H=1 TO 80
230 PRINT PIC$(V,H);
240 NEXT H:NEXT V
250 LOCATE ROW,COL
260 RETURN
```

OPEN-Befehl

Syntax:

OPEN <Modus>,[#]<Dateinummer>,
<Dateispez> [,<Satzlänge>]

OPEN <Dateispez>[(FOR-<Modus>] AS[#]
<Datei-nummer>[LEN=Satzlänge]

In der ersten Syntaxform entspricht "Modus" einem Zeichenfolgenausdruck, dessen erstes Zeichen eines der folgenden Zeichen darstellt:

O Gibt den sequentiellen Ausgabemodus an.

I Gibt den sequentiellen Eingabemodus an.

R Gibt den direkten Ein-/Ausgabe-Modus an.

Bei der zweiten Syntaxform ist Modus kein Zeichenfolgenausdruck (keine Anführungszeichen), sondern sieht folgendermaßen aus:

OUTPUT Gibt den sequentiellen Ausgabemodus an.

INPUT Gibt den sequentiellen Eingabemodus an.

APPEND Gibt den sequentiellen Ausgabemodus an und setzt den Dateizeiger an das Dateiende. Die Satznummer entspricht dem letzten Satz der Datei. Die Datei wird dann mit einem PRINT#- oder WRITE#-Befehl erweitert (angefügt).

Beachten Sie, daß Modus und Dateispezifikation bei der ersten Syntaxform in Anführungszeichen stehen. Bei der zweiten Form benötigt nur die Dateispezifikation Anführungszeichen. Wird in einer der Syntaxformen der Modus weggelassen, so geht GW-BASIC vom Direktzugriff aus. Sequentielle und Direktzugriffs-Dateien werden in Kapitel 5 "Dateien und Einheiten" beschrieben.

Unabhängig davon, welche Syntaxform benutzt wird, ist <Dateinummer> ein ganzzahliger Ausdruck, dessen Wert zwischen 1 und der Höchstzahl liegt, die beim Laden von GW-BASIC mit der Option /F festgelegt wurde. Diese Nummer ist dann mit der Datei verknüpft, solange sie eröffnet ist. Mit ihr nehmen andere E/A-Befehle auf die Datei Bezug, beispielsweise FIELD bei einer Direktzugriffsdatei.

<Dateispez> ist ein Zeichenfolgenausdruck, der einen Namen enthält, der den DOS-Regeln für Dateinamen gerecht wird. <Dateispez> kann einen Pfadnamen enthalten (siehe Kapitel 5).

<Satzlänge> ist ein ganzzahliger Ausdruck, mit dem, sofern er angegeben wird, die Satzlänge für Direktzugriffs- und sequentielle Dateien festgelegt wird. Die Standardlänge beträgt 128 Bytes. Kein Wert darf größer sein als der Wert, der beim Laden von GW-BASIC mit der Option /S festgelegt wurde.

Verwendung: Ermöglicht die Ein-/Ausgabe in einer Datei oder Einheit.

Bemerkungen: Eine Diskettendatei muß eröffnet werden, bevor eine E/A-Operation mit dieser Datei ausgeführt werden kann. Durch OPEN wird ein Puffer für die Ein-/Ausgabe in die Datei oder Einheit zugewiesen, und die Zugriffsart festgelegt, die mit dem Puffer benutzt wird.

Die GW-BASIC Befehle, bei denen die Datei vorher eröffnet werden muß, sind: FIELD, GET, INPUT$, INPUT#, LINE INPUT#, PRINT#, PRINT# USING, PUT und WRITE#.

Bei einer Diskettendatei kann es sich entweder um eine Direktzugriffsdatei oder eine sequentielle Datei handeln. Dasselbe gilt für einen Drucker. Auf andere Einheiten wird normalerweise im sequentiellen Modus zugegriffen. APPEND gilt nur für Diskettendateien.

Es sei denn, in <Dateispez> wird ein anderes Diskettenlaufwerk oder eine andere Einheit angege-

ben, geht GW-BASIC davon aus, daß die zu eröffnende Datei in dem aktuellen Diskettenlaufwerk liegt.

Das Eröffnen einer Datenübertragungsdatei wird separat beschrieben (siehe OPEN COM).

Eine Datei kann für die sequentielle Eingabe oder den Direktzugriff unter mehr als einer Dateinummer gleichzeitig eröffnet werden. Ist eine Datei jedoch eröffnet, so kann sie nicht auch noch für die sequentielle Ausgabe oder das Anfügen eröffnet werden.

Wird ein Drucker (LPT1:, LPT2: oder LPT3:) als Direktzugriffsdatei mit einer Breite von 255 eröffnet, so wird der Zeilenvorschub, der normalerweise eine Zeilenschaltung begleitet, unterdrückt. Der Druckkopf kann dann ein zweites Mal über die Zeile gehen und sie beispielsweise zusätzlich unterstreichen.

Beispiele:

Der folgende Befehl ist ein Beispiel für das Eröffnen einer neuen Direktzugriffsdatei:

```
10 OPEN "WÖRTER" FOR OUTPUT AS #1
```

Mit der anderen Syntaxform sieht dieser Befehle folgendermaßen aus:

```
10 OPEN "0", #1, "WÖRTER"
```

Diese Befehle dürfen nicht für eine schon vorhandene Datei benutzt werden, da diese Datei gelöscht und dann als neue Datei eröffnet würde. Statt dessen wird folgender Befehl benutzt:

```
20 OPEN "WÖRTER" FOR APPEND AS #1
```

Mit dem folgenden Beispiel wird eine Direktzugriffsdatei eröffnet. Steht die Datei STORY.TIM in Laufwerk C schon in dem gegenwärtigen Inhaltsverzeichnis, so wird sie zur weiteren Verarbeitung durch Ihr Programm eröffnet (sie wird nicht gelöscht). Ist die neue Datei noch nicht vorhanden, so wird sie als neue, leere Datei eröffnet. Eine Satzlänge von 256 Bytes wird angegeben.

```
10 OPEN "C:STORY.TIM" AS #1 LEN=256
```

Mit der anderen Syntaxform sieht dieser Befehl folgendermaßen aus:

```
10 OPEN "R", #9, "C:STORY.TIM", 256
```

Nachfolgend ein Beispiel für die Angabe eines Pfadnamens:

```
30 OPEN "BITOPLVL \ BOTLVL \ INFO.OLD"
   FOR APPEND AS 2
```

Hinweis:

Beim Eröffnen einer Datei kann es zu folgenden Fehlersituationen kommen:

"File not found" (Datei nicht gefunden)
Eine für die Eingabe eröffnete Datei ist nicht vorhanden. Ist eine für die Ausgabe, das Anfügen oder den Direktzugriff eröffnete Datei nicht vorhanden, so wird eine neue Datei erstellt.

"Illegal function call" (Unzulässiger Funktionsaufruf)
Ein Wert in dem OPEN-Befehl liegt außerhalb des zulässigen Bereichs.

Für "Dateispez" kann eine Zeichenfolgenvariable angegeben werden. Zum Beispiel:

```
10 INPUT "An welche sequentielle Datei
   müssen Daten angefügt werden";F$
20 OPEN F$ FOR APPEND AS 1
```

OPEN "COM-Befehl

Syntax:

OPEN "<Einh>:[<Geschwindigkeit>],
[<Parität>] , [<Daten>] [<Stop>] [,RS] [,CS [n]]
[,DS [n]] [, CD [n] [,LF] [,PE] " AS [#]
Dateinummer [LEN=Länge]

Verwendung:

Eröffnet eine Datenübertragungsdatei. Weist den Puffer für die Ein-/Ausgabe auf dieselbe Art und Weise wie OPEN bei Diskettendateien zu. Unterstützt die asynchrone Datenübertragung über RS-232 mit anderen Computern und Peripheriegeräten.

Bemerkungen:

<Einh>
Gibt eine der folgenden Datenübertragungseinheiten an: COM1 oder COM2.

<Geschwindigkeit>
Eine ganzzahlige Konstante, mit der angegeben wird, wie viele Bits pro Sekunde gesendet oder empfangen werden (Baudrate). Gültige Geschwindigkeiten sind: 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 und 9600. Der Standardwert lautet 300 Bit/s.

<Parität>
Eine aus einem Zeichen bestehende Konstante, mit der die Parität für das Senden und Empfangen wie folgt angegeben wird:

S — SPACE: Das Paritätsbit wird stets als Leerzeichen gesendet und empfangen (0-Bit).

O — ODD: Beim Senden und Empfangen wird auf ungerade Parität geprüft.

M — MARK: Das Paritätsbit wird stets als Marke gesendet und empfangen (1-Bit).

E — EVEN: Beim Senden und Empfangen wird stets auf gerade Parität geprüft.

N NONE: Beim Senden oder Empfangen findet keine Paritätsprüfung statt.

Standardmäßig wird eine gerade Parität (E) benutzt.

<Daten>
Eine ganzzahlige Konstante, die die Anzahl von gesendeten oder empfangenen Datenbits angibt. Gültige Werte sind 4, 5, 6, 7 und 8. Der Standardwert lautet 7. Geben Sie 4 an, so können Sie N nicht für die Parität angegeben, ansonsten kommt es zu einer Fehlermeldung "Bad File Name" (Falscher Dateiname). Werden 8 Bits angegeben, so muß N (keine) für die Parität angegeben werden.

<Stop>
Eine ganzzahlige Konstante, mit der die Anzahl von Stoppbits angegeben wird. Gültige Werte sind 1 und 2. Der Standardwert für die Stoppbits bei einer Baudrate von 75 oder 110 Bits/s lautet 2. Bei allen anderen Baudraten lautet der Standardwert 1. Wird 4 oder 5 für die Daten angegeben, so ergibt eine Eingabe von 2 für die Stoppbits 1 1/2 Stoppbits.

RS
Unterdrückt das RTS-Leitungssignal (Sendeanforderung). Wird RS aufgenommen, so wird die RTS-Leitung nicht eingeschaltet, wenn ein OPEN COM Befehl ausgeführt wird.

CS<n>
Steuert das CTS-Leitungssignal (Sendebereitschaft). Wird CS ohne n angegeben, so wird das Leitungssignal nicht überprüft. Wird CSn eingegeben, so wird mit n die Zeit (in Millisekunden) angegeben, während der das System wartet, bevor eine Fehlermeldung "Device Timeout" (Zeitsperre bei der Einheit) ausgegeben wird. Wird n = 0 benutzt, so ist dies gleichbedeutend mit der Eingabe von CS ohne n.

DS<n>
Steuert das DSR-Leitungssignal (Betriebsbereitschaft). Wird DS angegeben, so wird das Leitungssignal nicht überprüft. Wird DSn eingegeben, so wird mit n die Zeit (in Millisekunden) angegeben, während der das System wartet, bevor eine Fehlermeldung "Device Timeout" (Zeitsperre bei der Einheit) zurückgegeben wird. Wird n = 0 benutzt, so ist dies gleichbedeutend mit der Eingabe von DS ohne n.

CD<n>
Steuert das CD-Leitungssignal, das auch als RLSD-Leitungssignal bekannt ist (Empfangssignalpegel). Wird CD eingegeben, so wird das Leitungssignal nicht überprüft. Wird CDn eingegeben, so wird mit n die Zeit (in Millisekunden) angegeben, während der das System wartet, bevor eine Fehlermeldung "Device Timeout" (Zeitsperre bei der Einheit) zurückgegeben wird. Wird n = 0 benutzt oder die Option weggelassen, so wird das Leitungssignal nicht überprüft.

Der Höchtswert für n, der mit CS, DS oder CD angegeben werden kann, beträgt 65535.

LF
Sendet einen Zeilenvorschub im Anschluß an jede Zeilenschaltung. LF wird angegeben, wenn Datenübertragungsdateien für das Ausdrucken auf einem seriellen Zeilendrucker benutzt werden. Werden die Befehle INPUT# und LINE INPUT# dazu benutzt, Daten aus einer Datenübertragungsdatei zu lesen, die mit der Option LF eröffnet wurde, so ignorieren diese Befehle den Zeilenvorschub und stoppen, wenn sie eine Zeilenschaltung entdecken.

PE
Aktiviert die Paritätsprüfung. Wird dieser Parameter nicht angegeben, so findet keine Paritätsprüfung statt. Angenommen, es werden sieben Datenbits oder weniger benutzt und die Paritätsprüfung ist aktiviert, so setzt ein Paritätsfehler das höchstwer-

tige Bit und führt zu einer Fehlermeldung "Device I/O Error" (E/A-Fehler bei der Einheit). (Rahmen- und Überlauf-Fehler setzen unabhängig von der Datenlänge stets das höchstwertige Bit und führen zu der Fehlermeldung "Device I/O Error".

<Dateinummer>
Ein ganzzahliger Ausdruck, der eine gültige Dateinummer zurückgibt. Die Nummer ist mit der Datei verknüpft, solange sie eröffnet ist. Mit ihr nehmen andere E/A-Befehle zur Datenübertragung Bezug auf die Datei. Eine Datenübertragungseinheit kann nur unter einer Dateinummer gleichzeitig eröffnet sein.

<Länge>
Die Höchstzahl von Bytes, die bei Benutzung von GET oder PUT aus dem Datenübertragungspuffer gelesen werden kann. Der Standardwert lautet 128 Bytes.

Ein Syntaxfehler in dem in Anführungszeichen stehenden Teil des Befehls führt zu einer Fehlermeldung "Bad File Name" (Falscher Dateiname). Es wird nicht angegeben, welcher Parameter fehlerhaft ist.

Für Informationen über die Ein-/Ausgabe bei der Datenübertragung wird auf Kapitel 5 "Dateien und Einheiten" verwiesen. Eine erfolgreiche Datenübertragung hängt zu einem großen Teil von den richtigen Hardwareanschlüssen ab. Aus diesem Grund sollte die Hardware-Dokumentation für die benutzte Datenübertragungseinheit gelesen werden.

Beispiele:

In dem folgenden Beispiel wird Dateinummer 1 für die Datenübertragung mit folgenden Standardwerten eröffnet: 300 Bit/s, gerade Parität und 7 Datenbits mit 1 Stoppbit. Die Paritätsüberprüfung findet jedoch nicht wirklich statt, da PE nicht aufgenommen ist.

```
10 OPEN "COM1:" AS #1
```

Mit dem folgenden Befehl wird Dateinummer 2 für die Datenübertragung mit einer Baudrate von 2400 Bit/s eröffnet. Die Standardwerte lauten: gerade Parität, 7 Datenbits und 1 Stoppbit. Auch hier wird PE nicht angegeben.

```
10 OPEN "COM1:2400" AS #2
```

Mit dem folgenden Befehl wird Dateinummer 1 für die asynchrone E/A mit einer Baudrate von 1200 Bit/s eröffnet. Keine Parität wird erzeugt oder geprüft. Aus 8 Bits bestehende Bytes werden gesendet und empfangen. Standardmäßig wird ein Stoppbit benutzt.

```
10 OPEN "COM2:1200,N,8" AS #1
```

Mit dem nächsten Beispiel wird COM1 für eine Baudrate von 4800 Bit/s eröffnet, wobei standardmäßig die gerade Parität und 7 Datenbits benutzt werden. RTS muß gesendet werden. CTS wird nicht überprüft. Wird DSR nicht innerhalb von drei Sekunden ertkannt, so kommt es zu einer Fehlermeldung "Device Timeout" (Zeitsperre bei der Einheit). Die Paritätsprüfung ist aktiviert. Für die fehlenden (Standard-)-Parameter sind Kommas erforderlich: Parität, Daten und Stopp. (Werden einer oder mehrere der Parameter RS, CS, DS, CD, LF und PE weggelassen, so sind statt dessen keine Kommas erforderlich.)

```
10 OPEN "COM1:4800,,,,CS,DS3000,PE" AS #1
```

Hinweis:

Eine Unterbrechung bei "Device Timeout" (Zeitsperre bei der Einheit) wird empfohlen, wenn der Datenübertragungseinheit mehr Zeit für die Antwort zur Verfügung gestellt werden soll (siehe ON ERROR). Möglicherweise möchten Sie jedoch nicht unendlich auf diese Einheit warten. In diesem Fall reicht es nicht aus, wenn die Unterbrechungsroutine nur einen RESUME-Befehl enthält. Das Programm muß einen Zähler enthalten, mit dem die vorzunehmenden Wiederholversuche begrenzt werden. Die Unterbrechungsroutine kann diesen

Zähler immer dann anpassen, wenn sie von GW-BASIC ausgeführt wird. Zum Beispiel:

```
 10 ATTEMPT% = 5
 20 ON ERROR GOTO 500
 30 OPEN "COM1:,,,, CS,DS,CD5000" AS #1
  .
  .
  .
500 IF ERR <>24 THEN GOTO 600
510 ATTEMPT%=ATTEMPT%-1
520 IF ATTEMPT%=0 THEN GOTO 540
530 RESUME
540 BEEP:PRINT "Einheit prüfen und
    Programm erneut starten"
550 STOP
  .
  .
  .
600 REM Andere Fehlerbehandlungsroutinen
```

OPTION BASE-Befehl

Syntax: OPTION BASE n

wobei n gleich 1 oder 0 ist.

Verwendung: Deklariert den Mindestwert für Indexgrenzen.

Bemerkungen: Der Grundwert ist 0. Wird der Befehl

OPTION BASE 1

ausgeführt, so kann der niedrigste Wert einer Indexgrenze 1 sein.

GW-BASIC muß OPTION BASE ausführen, bevor eine Matrix definiert oder benutzt wird.

OUT-Befehl

Syntax: OUT I,J

I ist die Anschluß-Nummer. Hier muß es sich um einen ganzzahligen Ausdruck in dem Bereich von 0 bis 65535 handeln.

J entspricht den zu übertragenden Daten. Hier muß es sich um einen ganzzahligen Ausdruck in dem Bereich von 0 bis 255 handeln.

Verwendung: Sendet ein Byte an die Ausgabe-Anschlußposition des Rechners.

Beispiel: 100 OUT 128,255

Sendet den Wert 255 über Anschlußposition 128.

Hinweis: OUT hat dieselbe Auswirkung wie die OUT-Instruktion in der Assemblersprache.

PAINT-Befehl

Syntax: PAINT (x,y) [[<Effekt>] [,<Umriß>] [Hintergrund]

x,y
Die Koordinaten eines beliebigen Punktes innerhalb des auszufüllenden Bereichs. Die Koordinaten können in der absoluten oder relativen Form (mit STEP) angegeben werden.

<Effekt>
Handelt es sich bei diesem Parameter um einen numerischen Ausdruck, so kann er entweder die Hintergrundfarbe (0) oder eine der Farben 1 bis 3 aus der aktuellen Farbpalette (siehe COLOR) angeben. Dies gilt für Grafiken mit mittlerer Auflösung, bei denen der Standardwert 3 lautet. Bei einfarbiger Grafik mittlerer und hoher Auflösung gibt ein numerischer Wert von 0 die Farbe schwarz an; der Standardwert 1 gibt weiß an. "Effekt" kann jedoch auch einen Zeichenfolgenausdruck darstellen. In diesem Fall wird ein Überzeichnen mit einem Muster vorgenommen. Diese Funktion wird später in dieser Beschreibung von PAINT erläutert.

<Umriß>
Die Farbe für den Umriß des auszufüllenden Bereichs. Die möglichen Farben werden wie unter <Effekt> beschrieben angegeben. Ist die Farbe für den <Umriß> nicht richtig, so wird über den Bereich hinausgezeichnet, in dem x,y steht. Bei einfarbiger Grafik mittlerer und hoher Auflösung braucht <Umriß> nicht angegeben zu werden, da hier derselbe Wert wie bei <Effekt> benutzt wird.

Verwendung: Mit diesem Befehl wird bei einfarbiger Grafik mittlerer und hoher Auflösung ein Bereich mit einer angegebenen Farbe ausgefüllt. Bei Farbgrafiken mit niedriger und hoher Auflösung wird mit diesem Befehl ein schwarz eingerahmter Bereich schwarz ausgefüllt oder ein weiß eingerahmter Bereich weiß ausgefüllt.

Bemerkungen:

Der Anfangspunkt x,y muß innerhalb der zu zeichnenden Figur liegen und vollständig von ihr eingeschlossen sein. Hat der angegebene Punkt schon die "Umriß"-Farbe, so findet kein Auszeichnen statt. Deshalb bewegen Sie x,y einfach nach dem Zeichnen mit DRAW und vor dem Auszeichnen mit PAINT in die Figur.

Wird eine Figur mit vielen Ecken mit PAINT gezeichnet, so braucht GW-BASIC mehr als den üblichen Stack-Platz. Mit dem CLEAR-Befehl kann GW-BASIC mehr Stack-Platz zugewiesen werden.

Überzeichnen mit einem Muster

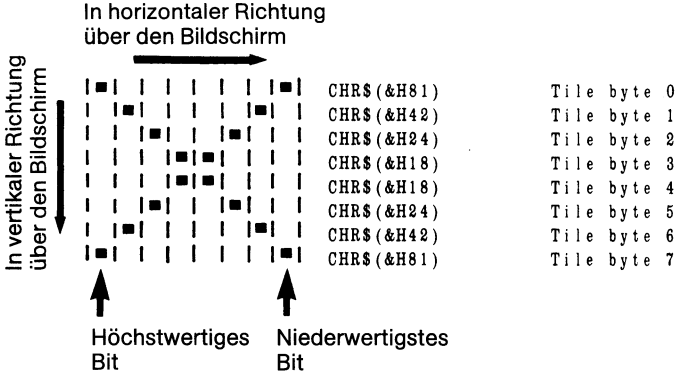
Das Überzeichnen ist eine Einrichtung von GW-BASIC, mit der Ihr Programm den gesamten oder einen Teil des Bildschirms mit einem Muster überzeichnen kann. Die Maske, mit der dieses Muster festgelegt wird, hat stets eine Breite von 8 Bits, so daß vier horizontale Bildelemente bei Farbgrafiken mit niedriger und hoher Auflösung oder 8 horizontale Bildelemente bei einfarbiger Grafik mit mittlerer und hoher Auflösung abgedeckt werden.

Ist <Effekt> ein Zeichenfolgenausdruck, so weiß GW-BASIC, daß Sie mit dem PAINT-Befehl ein Überzeichnen vornehmen möchten. In diesem Fall besteht <Effekt> aus bis zu 64 Hexadezimalwerten aus jeweils zwei Zeichen, von denen jeder ein Muster für vier oder acht horizontale Bildelemente darstellt. Auf diese Weise kann ein Muster mit bis zu 4 x 64 Bildelementen in horizontaler bzw. vertikaler Richtung bei Farbgrafiken mit niedriger und hoher Auflösung oder mit 8 x 64 Bildelementen bei einfarbiger Grafik mittlerer und hoher Auflösung definiert werden. Dieses Muster wird gleichmäßig über den gesamten Bildschirm oder innerhalb einer Figur wiederholt, wenn diese Figur den Punkt x,y umfaßt.

Bei einfarbiger Grafik mittlerer und hoher Auflösung führt der folgende Zeichenfolgenwert für "Effekt" zu einer Anzeige von x über den gesamten Bildschirm wobei ein 8 x 8 Pixel-Muster benutzt wird.

```
10 SCREEN 2:CLS
20 PAINT (320,100),CHR$(&H81)+
  CHR$(&H42)+CHR$(&H24)+
    CHR$(&H18)+CHR$(&H18)+
    CHR$(&H24)+CHR$(&H42)+CHR$(&H81)
```

Um zu verstehen, wie diese Hexadezimalwerte ein x erstellen, sind einige Kenntnisse der Binär- und Hexadezimalwerte nützlich. Vielleicht hilft das folgende Diagramm:



Der erste CHR$-Wert im Anschluß an die x,y-Koordinaten in Zeile 20 entspricht dem Überzeichnen-Byte 0, der letzte Wert entspricht dem Überzeichnen-Byte 7.

GW-BASIC startet nicht unbedingt mit Überzeichnen-Byte 0 bei den angegebenen x,y-Koordinaten. Statt dessen enspricht die Nummer des Überzeichnen-Bytes, das bei x,y gezeichnet wird, dem Rest der Division der y-Koordinate durch die Anzahl von Bytes in dem Muster. In dem vorliegenden Beispiel bedeutet dies, daß mit dem Zeichnen des Musters in der vertikalen Bildschirmzeile 100 mit Byte Nr. 4 begonnen wird (Rest der Division von 100 durch 8).

Bei Farbgrafiken mit niedriger und hoher Auflösung stellt ein Überzeichnen-Byte nur vier Bildschirmpunkte in horizontaler Richtung dar. Der Grund dafür liegt darin, daß für jeden Punkt zwei Bits erforderlich sind, so daß GW-BASIC weiß, welche Farbe dieser Punkt hat. Deshalb müssen die 8 Bits jedes Überzeichnen-Bytes als vier Paare betrachtet werden. Der Binärwert eines Paares

kann 01 bei Farbe 1 der Farbpalette, 10 bei Farbe 2 oder 11 bei Farbe 3 betragen. Dadurch wird die Farbe für einen einzelnen Bildschirmpunkt erhalten. Aus diesem Grund werden Sie wahrscheinlich die drei folgenden Hexadezimalwerte für die CHR$-Funktionen benutzen, aus denen die Zeichenfolge für "Effekt" bestehen soll:

&H55	Erzeugt vier fortlaufende horizontale Punkte in Farbe 1 (Palette 0: grün, Palette 1: kobaltblau).
&HAA	Erzeugt vier fortlaufende horizontale Punkte in Farbe 2 (Palette 0: rot, Palette 2: violett).
&HFF	Erzeugt vier fortlaufende horizontale Punkte in Farbe 3 (Palette 0: braun, Palette 1: weiß).

Unter Umständen haben Sie die Möglichkeit, einen schon gezeichneten Bereich mit einem Muster zu überzeichnen, der dieselbe Farbe wie zwei aufeinanderfolgende Bytes in einem Muster aufweist. Normalerweise bricht PAINT die Aus-führung ab, wenn es auf zwei aufeinanderfolgende Bytes mit derselben Farbe wie dem festgelegten Punkt stößt, da der Punkt dann mit derselben Farbe umgeben wird. Mit "Hintergrund" wird eine Hintergrund-Farbe für das Überzeichnen mit einem Muster angegeben, mit der Sie bis zu zwei aufeinanderfolgende Bytes in der Zeichenfolge für das Überzeichnen überspringen können. Wird <Hintergrund> benutzt, so kann das Programm beispielsweise abwechselnd blaue und rote Linien auf einem roten Hintergrund mit einem Mindestaufwand an GW-BASIC Befehlen zeichnen.

Stimmen mehr als zwei aufeinanderfolgende Linien in dem Muster mit "Hintergrund" überein, so erkennt GW-BASIC einen Fehler "Illegal function call" (Unzulässiger Funktionsaufruf).

Beispiele:

Mit dem ersten Beispiel wird ein Kreis in roter Farbe auf schwarzem Bildschirmhintergrund

gezeichnet. Da der letzte Punkt, auf den Bezug genommen wird, dann die Mitte des Kreises ist, befindet sich PAINT schon innerhalb des Kreises und braucht nicht bewegt zu werden: STEP (0,0). Der Kreis wird dann mit grüner Farbe (Farbe 1 der Palette 0) ausgefüllt. Die <Umriß>-Farbe muß auf 2 festgelegt werden, d.h. auf die Farbe, mit der der Kreis gezeichnet wurde, ansonsten wird außerhalb des Kreises weitergezeichnet.

```
10 SCREEN 1:COLOR 0,0
20 CLS
30 CIRCLE (160,100),30,2
40 PAINT STEP (0,0),1,2
```

Das nächste Beispiel benutzt einen Zeichenfolgenausdruck für <Effekt>, um kobaltblaue und violette horizontale Streifen in das Kästchen zu zeichnen, das mit Zeile 40 gezeichnet wird. Mit STEP (5,-5) wird die Position für PAINT von der unteren linken Ecke des Kästchens in den Kasten bewegt. Da das Kästchen in weißer Farbe gezeichnet wurde, muß "Umriß" auf weiß (3) gesetzt werden, da ansonsten über die Grenzen des Kästchens hinaus gezeichnet würde.

```
10 SCREEN 1.COLOR 0,1
20 CLS
30 DRAW "c3"
40 DRAW "u50 r50 d50 150"
50 PAINT STEP (5,-5),
   CHR$(&H55)+CHR$(&H55)+CHR$
   (&HAA)+CHR$(&HAA),3
```

Mit dem folgenden Beispiel wird die Benutzung von <Hintergrund> dargelegt, um kobaltblaue und violette Streifen auf einem violetten Hintergrund zu zeichnen. Mit den Zeilen 30 und 40 wird ein Kästchen gezeichnet und violett ausgefüllt. Mit Zeile 60 werden die Streifen gezeichnet, wobei der durch <Hintergrund> ermöglichte Überspring-Effekt benutzt wird. Zeile 70 enthält denselben Befehl als Bemerkung (REM) des Programmierers, jedoch ohne dieses Überspringen. Versuchen Sie, Zeile 60 zu der REM-Zeile (der nicht ausgeführten Zeile) zu

machen und lassen Sie GW-BASIC statt dessen Zeile 70 ausführen. Sie werden feststellen, daß das Zeichnen gestoppt wird, sobald die erste violette Linie gezeichnet werden soll. Beachten Sie, daß das Muster für das Überzeichnen in diesem Beispiel aus der Zeichenfolgenvariablen PATT$ genommen wurde. Dank der in Zeile 50 erzeugten Verzögerung können Sie beobachten, was auf dem Bildschirm geschieht.

```
10 SCREEN 1:COLOR 0,1
20 PATT$=CHR$(&H55)+CHR$(&H55)+
   CHR$(&HAA)+CHR$(&HAA)
30 DRAW "u90 r90 d90 190"
40 PAINT STEP (5,-5),2,3
50 FOR DLY%=1 TO 1000:NEXT DLY%
60 PAINT STEP (5,-5),PATT$,3,CHR$(&HAA)
70 REM PAINT STEP (5,-5),PATT$,3
```

PEEK-Funktion

Syntax: PEEK(I)

Verwendung: Gibt das aus dem angegebenen Speicherplatz (I) gelesene Byte zurück.

Bemerkungen: Der zurückgegebene Wert ist eine Ganzzahl in dem Bereich von 0 bis 255. I muß in dem Bereich von 0 bis 65535 liegen. I stellt den Abstand von dem aktuellen Segment dar, das mit dem letzten DEF SEG Befehl definiert wurde.

PEEK ist die Ergänzungsfunktion zu dem POKE Befehl.

Beispiel: A% = PEEK(&H5A00)

Weist den Wert des Bytes bei der Hexadezimaladresse 5A00 der Ganzzahlvariablen A% zu.

Für I braucht kein Hexadezimalwert benutzt zu werden. Das Dezimaläquivalent dieses Befehls lautet:

A% = PEEK(23040)

Hinweis: Soll einem bestimmten Speicherplatz ein Wert zugewiesen werden, so benutzen Sie den POKE-Befehl.

PEN-Befehl

Syntax:

PEN ON
PEN OFF
PEN STOP

Verwendung:

Aktiviert und deaktiviert das Lesen über den Lichtstift.

Bemerkungen:

Der Lichtstift ist ursprünglich ausgeschaltet. Mit PEN ON werden Daten von dem Lichtstift mit der PEN-Funktion gelesen. Die Programmunterbrechung ON PEN wird aktiviert.

Mit PEN OFF werden das Lesen durch den Lichtstift und die Programmverzweigung ON PEN deaktiviert. Dieser Befehl wird ausgegeben, sobald keine Daten mehr über den Lichtstift eingelesen werden müssen, da dies zu besseren Ausführungszeiten von GW-BASIC führt. In diesem Fall werden PEN-Aktivitäten nicht mehr von GW-BASIC aufgezeichnet.

Durch PEN STOP wird die Programmverzweigung ON PEN deaktiviert, GW-BASIC zeichnet jedoch sämtliche Aktivitäten des Lichtstiftes auf. Demzufolge wird bei Ausführung von PEN ON eine Programmverzweigung aktiviert, wenn es zwischenzeitlich zu einer Aktvitität des Lichtstiftes gekommen ist.

PEN-Funktion

Syntax: PEN(n)

n ist ein numerischer Ausdruck in dem Bereich von 0 bis 9, mit dem ein bestimmter zu lesender Wert des Lichtstiftes ausgewählt wird.

Bemerkungen: Die Werte 0 bis 9 haben folgende Bedeutung:

0 Hier handelt es sich um ein Flag, mit dem angegeben wird, ob der Schalter für den Lichtstift nach unten gestellt wurde, seit die PEN-Funktion das letzte Mal aufgerufen wurde.

1 Gibt die x-Koordinate der Position zurück, in der der Lichtstift das letzte Mal aktiviert wurde. Die so gelesene Zahl kann bei Grafiken mit niedriger Auflösung in dem Bereich von 0 bis 319 und bei Grafiken mit mittlerer und hoher Auflösung in dem Bereich von 0 bis 639 liegen.

2 Gibt eine Zahl (in niedriger und mittlerer Auflösung 0 bis 199, in hoher Auflösung 0 bis 399) zurück, die die y-Koordinate der Position darstellt, in der der Lichtstift als letztes gelesen wurde.

3 Gibt -1 zurück, wenn der Schalter für den Lichtstift nach unten gestellt ist. Ist er nach oben gestellt, so wird 0 angegeben.

4 Gibt die letzte bekannte gültige x-Koordinate zurück (0 bis 319 bei niedriger Auflösung, 0 bis 639 bei mittlerer und hoher Auflösung).

5 Gibt die letzte bekannte gültige y-Koordinate zurück (0 bis 199 in niedriger und mittlerer Auflösung, in hoher Auflösung 0 bis 399).

6 Gibt einen Wert im Bereich von 1 bis 24 für die Zeilenposition zurück, in der der Lichtstift das letzte Mal aktiviert wurde.

7 Gibt die Position der Zeichenspalte zurück, in der der Lichtstift das letzte Mal aktiviert wurde. Hier handelt es sich um einen Wert in dem Bereich von 1 bis 80 oder von 1 bis 40, je nach aktueller Einstellung der Bildschirmbreite.

8 Gibt die letzte bekannte gültige Zeilenposition in dem Bereich von 1 bis 24 zurück.

9 Gibt die letzte bekannte gültige Spaltenposition in dem Bereich von 1 bis 80 oder 1 bis 40 zurück, je nach aktueller Einstellung der Bildschirmbreite.

Beispiel:

```
10 PEN ON
20 PENLIN% = PEN(6)
```

aktiviert das Lesen über den Lichtstift (und die Programmverzweigung bei einer bestimmten Aktivität), und setzt die Nummer der Bildschirmzeile, in der der Lichtstift das letzte Mal aktiviert wurde, in die Variable PENLIN%.

Hinweis: Wird versucht, mit dem Lichtstift zu lesen, während PEN OFF gültig ist, so kommt es zu der Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

PLAY-Befehl

Syntax: PLAY <Zeichenfolgenausdruck>

Verwendung: Erstellt einen Ton, indem dessen Eigenschaften in dem Zeichenfolgenausdruck definiert werden. Der Ausdruck kann aus einem der folgenden Befehle bestehen, die Sie in beliebiger Reihenfolge angeben können, es sei denn, in der Beschreibung werden ausdrücklich andere Angaben gemacht.

A bis G [#,+,-]
Spielt die angegebene Note. # oder + im Anschluß an eine Note gibt Fis an. – im Anschluß an eine Note gibt B an. In beiden Fällen muß es sich bei der Note um eine echte Klaviertaste handeln.

L <n> – Länge
Legt die Länge der Note (bzw. Noten) fest, wobei n von 1 bis 64 gehen kann. So gibt L1 beispielsweise eine ganze Note, L2 eine halbe Note... und L64 eine 64tel-Note an. Sie können die Länge vor einer Gruppe von Noten oder im Anschluß an eine einzelne Note angeben, um nur die Länge dieser Note zu ändern. So entspricht A16 beispielsweise derselben Definition wie L16A.

MB – Musik im Hintergrund
Die Musik wird im Hintergrund gespielt. Ein Pufferspeicher aus bis zu 32 Noten spielt im Hintergrund, während GW-BASIC andere Befehle ausführt.

MF – Musik im Vordergrund
Die Musik wird im Vordergrund gespielt. Jede nachfolgende Note oder jeder nachfolgende Ton wird erst gestartet, wenn die vorhergehende Note bzw. der vorhergehende Ton beendet ist. MF ist der ursprüngliche Standardwert.

MN – Musik normal
Spielt jede Note mit 7/8 der in L (Länge) angegebenen Zeit. Hier handelt es sich um die Standardangabe.

ML – Musik Legato
Spielt jede Note mit voller Länge (wie in L angegeben).

MS – Musik Stakkato
Spielt jede Note mit 3/4 der in L (Länge) angegebenen Zeit.

N <n> – Note
Spielt die mit <n> angegebene Note. <n> kann von 0 bis 84 gehen und deckt so die Halbtöne mit sieben Oktaven ab, wobei zwei Oktaven unter dem mittleren C begonnen wird. Soll eine Pause angegeben werden, so kann <n> gleich 0 sein. Mit diesem Befehl kann die Note anders als mit Namen (A bis G) und Oktave angegeben werden.

O <n> – Oktave
Legt die Oktave fest, wobei <n> von 0 bis 6 gehen kann. Jede der sieben Oktaven beginnt mit C. Das mittlere C steht am Anfang von Oktave 3; die Standardoktave ist Oktave 4.

P <n> – Pause
Legt die Länge der Pause fest, wobei <n> von 1 bis 64 gehen kann. Der Wert <n> ist identisch mit dem Wert <n> in dem Befehl L (Länge); so veranlaßt P1 beispielsweise eine Pause während der Länge einer ganzen Note. P2 veranlaßt eine Pause während der Länge einer halben Note usw.

T <n> – Tempo
Legt die Anzahl von Viertelnoten <n> fest, die in einer Minute gespielt werden können. <n> kann von 32 bis 255 gehen. Der Standardwert lautet 120.

. – Punkt
Wird der Punkt im Anschluß an eine Note angegeben, so wird die Note als gepunktete Note gespielt, d.h. ihre Länge wird mit 3/2 multipliziert. Im Anschluß an die Note kann mehr als ein Punkt benutzt werden. In diesem Fall wird die Länge entsprechend angepaßt. So spielt A.. beispielsweise 9/4 mal so lange wie mit L angegeben, A... spielt 27/8

mal so lange usw. Punkte können auch nach einer Pause (P) benutzt werden, um die Pausenlänge auf dieselbe Art und Weise festzulegen.

X Variable;
Führt die angegebene Zeichenfolge mit gültigen PLAY-Befehlen aus.

> Note
Erhöht die Skala um eine Oktave und spielt die angegebene Note A bis G mit der neuen Oktave. Hat die Oktave schon den Wert 6, so wird die Note mit Oktave 6 gespielt.

< Note
Vermindert die Skala um eine Oktave und spielt die angegebene Note A bis G. Ist die Oktave schon gleich 0, so wird die Note mit Oktave 0 gespielt.

In allen Befehlen kann der Wert <n> eine konstante oder eine numerische Variable sein, vor der ein Gleichzeichen und hinter der ein Semikolon steht. Das Semikolon (;) ist erforderlich, wenn eine Variable auf diese Art und Weise benutzt wird, und wenn der Befehl X benutzt wird. Ansonsten ist ein Semikolon zwischen den Befehlen wahlweise. Im Anschluß an MF, MB, MN, ML oder MS ist es jedoch nicht zulässig. Leerzeichen in einer Zeichenfolge werden ignoriert.

Variablen können auch in der Form VARPTR$ (<Variable>) anstelle von =<Variable>; angegeben werden. Diese Methode ist bei Programmen nützlich, die später kompiliert werden.

Beispiele:

```
10 MARY$="GFE-FGGG"
20 PLAY "MB T100 03 L8;XMARY$;P8 FFF4"
30 PLAY "GB-B-4;XMARY$;GFFGFE-."
```

In dem folgenden Beispiel wird die Benutzung der Programmverzweigung ON PLAY dargestellt, um einen fortlaufenden Hintergrundton zu erzeugen, während eine Bildschirmaktivität ausgeführt wird. Die Bildschirmaktivität erstellt einfach ein Zufalls-

muster mit Leerzeichen, während die Leertaste betätigt wird (Zeilen 40 bis 60).

Mit Zeile 30 wird festgelegt, daß sich GW-BASIC ab Zeile 160 zu der Subroutine verzweigt, wenn die Anzahl von noch zu spielenden Noten für die Hintergrundmusik von 1 auf 0 geht. Mit der Subroutine wird GW-BASIC angewiesen, den Hintergrundton (erneut) zu spielen. Mit Zeile 40 wird die Programmverzweigung aktiviert. Zeile 50 spielt den Hintergrundton mit dem Befehl PLAY. Ohne diesen Befehl würde die Bedingung für eine Programmverzweigung bei Musik, die in Zeile 30 angegeben wird, niemals erfüllt, und es würde keine Musik gespielt.

```
 10 SCREEN 0:WIDTH 80
 20 KEY OFF
 30 ON PLAY (1) GOSUB 160
 40 PLAY ON
 50 GOSUB 160
 60 CLS
 70 IF INKEY$="" THEN GOTO 70
 80 COLOR INT(32*RND)
 90 LI%=INT(25*RND+1)
100 PO%=INT(80*RND+1)
110 IF LI%>25 THEN LI%=24
120 IF PO%>80 THEN PO%=80
130 LOCATE LI%,PO%
140 PRINT CHR$(219);
150 GOTO 70
160 REM *********
170 PLAY "o2 mb t140 f aa c aa"
180 RETURN
```

PMAP-Funktion

Syntax: PMAP (<Koord.>,<Übersetzung>)

Verwendung: Übersetzt die (mit WINDOW festgelegten) Weltkoordinaten in physikalische Koordinaten (siehe VIEW) und umgekehrt.

Bemerkungen: "Übersetzung" kann einen von vier Werten annehmen:

0 gibt eine physikalische x-Koordinate für eine in "Koord" angegebene Weltkoordinate x zurück.

1 gibt eine physikalische y-Koordinate für die in "Koord" angegebene Weltkoordinate y zurück.

2 gibt eine Weltkoordinate x für die in "Koord" angegebene physikalische x-Koordinate zurück.

3 gibt eine Weltkoordinate y für die in "Koord" angegebene physikalische y-Koordinate zurück.

Beispiel: Die physikalischen Koordinaten des benutzbaren Bildschirmbereichs liegen bei niedriger Auflösung zwischen 0,0 u.199,319 (bei mittlerer Auflösung bei 0,0-199,639 bei hoher Auflösung bei 0,0-399,639), solange diese Werte nicht mit VIEW geändert werden.

Wird als erstes WINDOW benutzt, um ein karthesisches Koordinatenschema bei Grafiken mit niedriger Auflösung festzulegen

```
10 SCREEN 1: WINDOW (-1,-1)-(1,1)
```

und wird dann:

```
20 PRINT PMAP(-1,0),PMAP(1,0),PMAP(-1,1),
   PMAP(1,1)
30 PRINT PMAP(0,1),PMAP(0,0)
```

ausgeführt, so zeigt GW-BASIC folgende Werte an:

```
0       319     199     0
100     160
```

Für weitere Angaben über die Festlegung des Koordinatenschemas mit diesem Befehl wird auf die Beschreibung von WINDOW verwiesen.

POINT-Funktion

Syntax:

POINT(x,y)
POINT(<Koord>)

Verwendung:

Liest die Farbe oder eine Koordinate des gerade adressierten Punktes auf dem Bildschirm.

Bemerkungen:

x und y müssen eine absolute Bildschirmposition angeben. Der bei Farbgrafiken mit niedriger und hoher Auflösung zurückgegebene Wert beträgt 0 bei der Hintergrundfarbe oder weist einen Wert 1, 2 oder 3 bei der entsprechenden Farbe der aktuellen Farbpalette auf. Bei einfarbigen Grafiken mittlerer und hoher Auflösung sind die möglichen Werte 0 und 1.

"Koord" ist ein Wert von 0 bis 3;

0 gibt die physikalische x-Koordinate zurück.

1 gibt die physikalische y-Koordinate zurück.

2 ist WINDOW gerade aktiv, so entspricht die zurückgegebene Koordinate der Weltkoordinate x. Ansonsten wird die physikalische Koordinate x zurückgegeben.

3 ist gleichbedeutend mit 2, allerdings wird eine y-Koordinate zurückgegeben.

Beispiele:

Mit dem folgenden Programm wird ein Zufallsmuster mit roten Punkten in einem 50 x 50 Bereich in der oberen linken Ecke des Bildschirms (Zeilen 20 bis 40) erstellt. Danach wird jeder der 2500 Grafikpunkte mit der POINT-Funktion gelesen. Ist ein Punkt rot, so wird er in schwarz geändert und umgekehrt.

```
10 SCREEN 1:COLOR 0,0:CLS
20 FOR CHANCE%=1 to 1000
30 PSET (RND*50,RND*50),2
40 NEXT CHANCE%
50 FOR X% = 0 TO 50
60 FOR Y% = 0 TO 50
70 PSET(X%,Y%),ABS(POINT(X%,Y%)-2)
80 NEXT Y%:NEXT X%
```

Mit dem nachfolgenden Programm werden Sie aufgefordert, eine x- und y-Koordinate einzugeben. Dieser Punkt wird dann mit dem Standardkoordinatensystem bei Grafiken mit niedriger Auflösung rot angezeigt (Ursprung oben links, 320 Punkte in horizontaler Richtung, 200 Punkte in vertikaler Richtung). Mit Zeile 40 wird dann ein karthesisches Koordinatensystem erstellt, wobei der Ursprung so nahe wie möglich bei der Mitte des Bildschirms liegt. Der Mittelpunkt wird braun angezeigt. Mit den von Ihnen eingegebenen x- und y-Werten wird ein grüner Punkt in Übereinstimmung mit dem neuen (karthesischen) Koordinatensystem gezeichnet. In der oberen Hälfte des Bildschirms werden dann auf der linken Seite die Koordinaten des grünen Punktes entsprechend dem physikalischen Bildschirm angezeigt. Auf der rechten Seite werden die Koordinaten desselben Punktes entsprechend der WINDOW-Definition angezeigt.

```
 10 INPUT "X and Y";X%,Y%
 20 SCREEN 1;COLOR 0,0:CLS
 30 PSET (X%,Y%),2
 40 WINDOW (-160,-100)-(159,99)
 50 PSET (0,0),3
 60 PSET (X%,Y%),1
 70 PRINT POINT(0);POINT(1);"--> WINDOW
    -->";POINT(2);POINT(3)
 80 IF INKEY$=""THEN 80
 90 SCREEN 0:WIDTH 80
100 LIST
```

POKE-Befehl

Syntax: POKE I,J

wobei I und J ganzzahlige Ausdrücke darstellen.

Verwendung: Schreibt ein Byte in eine Speicherposition.

Bemerkungen: I und J sind ganzzahlige Ausdrücke. Der Ausdruck I stellt die Adresse der Speicherposition dar. J entspricht dem Datenbyte. I muß in dem Bereich von 0 bis 65535 liegen. I ist der Abstand von dem aktuellen Segment, das mit dem letzten DEF SEG Befehl festgelegt wurde.

Die Ergänzungsfunktion zu POKE ist PEEK. Der Parameter bei PEEK ist eine Adresse, von der ein Byte gelesen werden muß.

Die POKE- und PEEK-Befehle ermöglichen die schnelle Speicherung und das schnelle Lesen von Daten bzw. die schnelle Übergabe von Daten in Routinen in der Maschinensprache. Mit POKE können sogar Routinen in der Maschinensprache geschrieben werden (siehe Kapitel 6).

Hinweis: Mit POKE erhalten Sie großen Einfluß auf den Computer, so daß dieser Befehl mit Vorsicht benutzt werden sollte. Es wird empfohlen, POKE ausschließlich in einem Speicherbereich zu benutzen, der speziell für Ihren Gebrauch reserviert ist.

POS-Funktion

Syntax: POS(I)

Verwendung: Gibt die aktuelle horizontale (Spalten-)-Position des Cursors zurück.

Bemerkungen: Die linksbündigste Position ist 1. I ist ein Pseudo-Parameter, so daß an seiner Stelle eine numerische Konstante benutzt werden kann. Mit der CSRLIN-Funktion wird die aktuelle Zeilenposition des Cursors zurückgegeben.

Beispiel: Durch

```
IF POS(X)>30 THEN BEEP
```

gibt GW-BASIC ein akustisches Signal (BEEP) aus, während der Cursor rechts von der dreißigsten Bildschirmspalte steht.

Hinweis: Der Bildschirm kann aus 40 oder 80 Spalten bestehen, je nach dem mit WIDTH fesgelegte Wert.

PRESET- und PSET-Befehle

Syntax:

PRESET(x,y) [,<Farbe>]
PSET(x,y) [,<Farbe>]

x und y geben einen Punkt auf dem Bildschirm an. "Farbe" gibt die Hintergrundfarbe (0) oder eine Farbe 1 bis 3 aus der aktuellen Farbpalette (bei Farbgrafiken mit niedriger und hoher Auflösung) an. Bei einfarbigen Grafiken mittlerer und hoher Auflösung geben die Werte 0 und 2 schwarz und 1 bzw. 3 weiß an.

Verwendung:

Zeigt einen Punkt auf dem Bildschirm in einer angegebenen Farbe an und/oder legt den Punkt fest, der von nachfolgenden Grafiken als der letzte Punkt betrachtet werden soll, auf den Bezug genommen wurde.

Die Standardfarbe für Farbgrafiken mit niedriger und hoher Auflösung ist 3 und für einfarbige Grafiken mittlerer und hoher Auflösung 1.

Der einzige Unterschied zwischen PSET und PRESET besteht darin, daß die Hintergrundfarbe (0) benutzt wird, wenn für PRESET keine "Farbe" angegeben wird. Auf diese Weise wird dann ein unsichtbarer Punkt gezeichnet.

Mit STEP können verschobene Koordinaten angegeben werden, d.h. ein Punkt relativ zu dem letzten Referenzpunkt.

Beispiel:

Das folgende Beispiel sendet einen Gedankenstrich, der als sechs horizontale Bildelemente definiert wird, von links nach rechts über die Mitte des Bildschirms. Der PRESET-Befehl in Zeile 70 bedeutet, daß kein Schweif übrig bleibt.

```
10 SCREEN 2:CLS
20 FOR X%= 0 TO 5
30 PSET (X%,100)
40 NEXT X%
50 FOR X%= 6 TO 639
60 PSET (X%,100)
```

```
70 PRESET (X%-6,100)
80 NEXT X%
```

Hier wird auch auf die praktischen Übungen am Ende von Kapitel 3 verwiesen.

Hinweis:

GW-BASIC erkennt keinen Fehler, wenn Sie versuchen, Punkte zu adressieren, die außerhalb des für das Zeichnen zur Verfügung stehenden Koordinatenbereichs liegen.

Wird ein größerer Wert als 3 für "Farbe" angegeben, so kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

PRINT-Befehl

Syntax: PRINT [<Liste mit Ausdrücken>]

Verwendung: Zeigt Daten auf dem Bildschirm an.

Bemerkungen: Wird <Liste mit Ausdrücken> weggelassen, so wird eine leere Zeile ausgedruckt. Wird "Liste mit Ausdrücken" angegeben, so werden die Werte der Ausdrücke auf dem Bildschirm angezeigt. Bei den Ausdrücken in der Liste kann es sich um numerische und/oder Zeichenfolgenausdrücke handeln. (Zeichenfolgen müssen in Anführungszeichen stehen.)

Druckpositionen

Die Position jedes gedruckten Elementes wird durch das Satzzeichen bestimmt, mit dem die Elemente in der Liste voneinander getrennt werden. GW-BASIC unterteilt die Zeile in Zonen mit jeweils 14 Leerzeichen. In der Liste mit Ausdrücken verursacht ein Komma, daß der nächste Wert am Anfang der nächsten Zone angezeigt wird. Bei einem Semikolon wird der nächste Wert unmittelbar im Anschluß an den letzten Wert angezeigt. Ein oder mehrere Leerzeichen zwischen den Ausdrükken haben dieselbe Auswirkung wie ein Semikolon.

Wird die Liste mit Ausdrücken durch ein Komma oder Semikolon beendet, so beginnt der nächste PRINT-Befehl mit der Anzeige auf derselben Zeile, wobei die Abstände entsprechend festgelegt werden. Wird die Liste mit Ausdrücken ohne ein Komma oder Semikolon beendet, so wird eine Zeilenschaltung am Ende der Zeile ausgeführt. Ist die anzuzeigende Zeile länger als die Bildschirmbreite, so geht GW-BASIC zu der nächsten physikalischen Zeile und fährt mit dem Rest der Zeile fort.

Auf gedruckte Zahlen folgt stets ein Leerzeichen. Vor positiven Zahlen steht ein Leerzeichen. Vor negativen Zahlen steht ein Minuszeichen. Zahlen mit einfacher Genauigkeit, die mit 6 oder weniger Ziffern in einem nichtskalierten Format (Fest-

punkt- oder Ganzzahl-Format) mit derselben Genauigkeit wie im skalierten Format (Gleitpunktformat) dargestellt werden können, werden im nichtskalierten Format ausgegeben. So wird 1 E-7 beispielsweise als .00000001 ausgegeben, 1E-8 wird jedoch als 1E-08 ausgegeben. Zahlen mit doppelter Genauigkeit, die mit 16 oder weniger Ziffern im nichtskalierten Format mit derselben Genauigkeit wie im skalierten Format dargestellt werden können, werden im nichtskalierten Format ausgegeben. So wird 1D-16 beispielsweise als .0000000000000001 ausgegeben, während 1D-17 als 1D-17 ausgegeben wird.

Anstelle des Wortes PRINT in einem PRINT-Befehl kann ein Fragezeichen benutzt werden. GW-BASIC ersetzt dieses Fragezeichen automatisch in der nächsten Auflistung mit LIST durch das Wort PRINT.

Beispiele:

```
10 X = 5
20 PRINT X+5,X-5,X*(-5),X^5
30 END
 ergibt
 10        0        -25        3125
```

In dem obigen Beispiel führen die Kommas in dem PRINT-Befehl dazu, daß jeder Wert am Anfang der nächsten Druckzone angezeigt wird.

In dem nächsten Beispiel führt das Semikolon am Ende von Zeile 20 dazu, daß die PRINT-Elemente von Zeile 20 und 30 auf derselben Zeile angezeigt werden. Aufgrund von Zeile 40 wird eine Leerzeile vor der nächsten Eingabeaufforderung ausgedruckt.

```
10 INPUT X
20 PRINT X"IM QUADRAT IST X^2 "UND";
30 PRINT X "HOCH DREI" X^3
40 PRINT
50 GOTO 10
ergibt
? 9
 9 IM QUADRAT IST 81 UND 9 HOCH DREI
   IST 729
```

```
? 21
 21 IM QUADRAT IST 441 UND 21 HOCH
    DREI IST 9261

?
```

In dem folgenden Beispiel führen die Semikolons in dem PRINT-Befehl dazu, daß jeder Wert unmittelbar im Anschluß an den vorhergehenden Wert angezeigt wird. (Vergessen Sie nicht, auf eine Zahl folgt stets ein Leerzeichen.) In Zeile 40 wird ein Fragezeichen anstelle des Wortes PRINT benutzt.

```
10 FOR X=1 TO 5
20 J=J+5
30 K=K+10
40 ?J;K;
50 NEXT X
ergibt
 5   10   10   20   15   30   20   40   25   50
```

Hinweis:

Wird das Ende des letzten Elementes eines PRINT-Befehls in der rechtsbündigsten Bildschirmposition (Spalte 40 oder 80 je nach WIDTH) angezeigt und wird dieser PRINT-Befehl nicht durch ein Semikolon beendet, so steht zwischen der gerade angezeigten Zeile und dem nächsten angezeigten Element eine Leerzeile.

Mit LPRINT werden Daten auf dem Drucker und nicht auf dem Bildschirm ausgedruckt.

PRINT USING-Befehl

Syntax: PRINT USING <Zeichenfolgenausdruck>;<Liste Ausdrücken>

Verwendung: Druckt Zeichenfolgen oder Zahlen mit einem bestimmten Format aus.

Bemerkungen/ Beispiele: <Liste mit Ausdrücken> besteht aus Zeichenfolgenausdrücken oder numerischen Ausdrücken, die durch Semikolon oder Komma getrennt ausgedruckt werden müssen.

<Zeichenfolgenausdruck> ist eine Zeichenfolgenkonstante oder Variable, die aus besonderen Formatierzeichen besteht. Diese Formatierzeichen (siehe unten) bestimmen das Feld und das Format der ausgedruckten Zeichenfolgen oder Zahlen.

Zeichenfolgenfelder

Wird PRINT USING für das Ausdrucken von Zeichenfolgen benutzt, so kann eines von drei Formatierzeichen für das Formatieren des Zeichenfolgenfeldes benutzt werden:

"!"
Gibt an, daß nur das erste Zeichen jedes Elementes in der "Liste mit Ausdrücken" ausgedruckt werden muß.

"\n Leerstellen\"
Gibt an, daß 2 + n Zeichen aus jedem Element der "Liste mit Ausdrücken" ausgedruckt werden müssen. Werden die umgekehrten Schrägstriche ohne Leerstellen eingegeben, so werden zwei Zeichen ausgedruckt; werden sie mit einer Leerstelle angegeben, so werden drei Zeichen ausgedruckt usw. Ist die Zeichenfolge länger als das Feld, so werden die zusätzlichen Zeichen ignoriert. Ist das Feld länger als die Zeichenfolge, so wird die Zeichenfolge linksbündig in dem Feld ausgerichtet und rechtsbündig mit Leerstellen aufgefüllt.

Beispiel:

```
10 A$="LOOK":B$="OUT"
30 PRINT USING "!";A$;B$
40 PRINT USING "[    ]";A$;B$
50 PRINT USING "[    ]";A$;B$;"!!"
ergibt
LO
LOOKOUT
LOOK OUT!!
```

"&"
Gibt ein Zeichenfolgenfeld mit variabler Länge an. Wird das Feld mit "&" angegeben, so wird die Zeichenfolge ohne Änderung ausgegeben.

Beispiel:

```
10 A$="LOOK";B$="OUT"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
ergibt
LOUT
```

Numerische Felder

Wird PRINT USING für das Ausdrucken von Zahlen benutzt, so können die folgenden Sonderzeichen für das Formatieren des numerischen Feldes benutzt werden:

Doppelkreuz
Ein Doppelkreuz wird zur Darstellung jeder Ziffernposition benutzt. Ziffernpositionen werden stets aufgefüllt. Verfügt die auszudruckende Zahl über weniger Stellen als Positionen angegeben wurden, so wird die Zahl in dem Feld rechtsbündig ausgerichtet (vor ihr stehen Leerzeichen).

. (Dezimalpunkt)
Ein Dezimalpunkt kann an beliebiger Stelle in dem Feld eingefügt werden. Gibt die Formatzeichenfolge an, daß vor dem Dezimalpunkt eine Ziffer stehen soll, so wird die Ziffer stets ausgedruckt (ggf. als Null). Zahlen werden nach Bedarf abgerundet.

```
PRINT USING "##.##";.78
0.78

PRINT USING "###.##";987.654
987.65

PRINT USING "##.##";10.2,5.3,66.789,.234
10.20    5.30    66.79    0.23
```

In dem letzten Beispiel wurden drei Leerstellen am Ende der Formatzeichenfolge eingefügt, um die ausgedruckten Werte auf der Zeile voneinander zu trennen.

+ (Pluszeichen)
Ein Pluszeichen am Anfang oder Ende der Formatzeichenfolge führt dazu, daß das Vorzeichen der Zahl (plus oder minus) vor oder hinter der Zahl ausgedruckt wird.

– (Minuszeichen)
Ein Minuszeichen am Ende des Formatfeldes führt dazu, daß negative Zahlen mit einem abschließenden Minuszeichen ausgedruckt werden.

```
PRINT USING "+##.##";-68.95,2.4,55.6,-.9
-68.95    +2.40    +55.60    -0.90

PRINT USING "##.##- ";-68.95,22.449,-7.01
68.95-    22.45    7.01-
```

** (Doppelte Sternchen)
Doppelte Sternchen am Anfang der Formatzeichenfolge geben an, daß führende Leerstellen in dem numerischen Feld mit Sternchen aufgefüllt werden. Die ** geben auch Positionen für zwei weitere Ziffern an.

```
PRINT USING "**#.#";12.39,-0.9,765.1
*12.4    *-0.9    765.1
```

$$ (Doppeltes Dollarzeichen)
Bei einem doppelten Dollarzeichen wird ein Dollarzeichen unmittelbar links von der formatierten

Zahl ausgedruckt. Das $$ gibt zwei weitere Ziffernpositionen an, von denen eine das Dollarzeichen darstellt. Mit $$ kann das Exponentialformat nicht benutzt werden. Negative Zahlen können nicht benutzt werden, es sei denn, das Minuszeichen steht rechts.

```
PRINT USING "$$###.##";456.78
$456.78
```

**$

Das **$ am Anfang einer Formatzeichenfolge kombiniert die beiden oben beschriebenen Symbole. Führende Leerstellen werden mit Sternchen aufgefüllt und vor der Zahl wird ein Dollarzeichen ausgedruckt. **$ gibt drei weitere Ziffernpositionen an, von denen eine das Dollarzeichen darstellt.

```
PRINT USING "**$##.##";2.34
***$2.34
```

, (Komma)

Steht ein Komma links von dem Dezimalpunkt in einer Formatierzeichenfolge, so wird links von jeder dritten Ziffer links vom Dezimalpunkt ein Komma ausgedruckt. Ein Komma am Ende der Formatzeichenfolge wird als Teil der Zeichenfolge ausgedruckt. Ein Komma gibt eine weitere Ziffernposition an. Das Komma hat keine Auswirkung, wenn es mit dem Exponentialformat (^^^^) benutzt wird.

```
PRINT USING "### #,.##";1234.5
1,234.50

PRINT USING "####.##,";1234.5
1234.50,
```

^^^^

Vier Auslassungszeichen können hinter die Zeichen der Ziffernposition gesetzt werden, um das Exponentialformat anzugeben. Die vier Auslassungszeichen lassen Platz für das Ausdrucken von

E+xx. Für den Dezimalpunkt kann eine beliebige Position angegeben werden. Die signifikanten Ziffern sind linksbündig ausgerichtet und der Exponent wird justiert. Es sei denn, ein führendes oder ab-schließendes + oder – wird angegeben, wird eine Ziffernposition links von dem Dezimalpunkt dazu benutzt, ein Leerzeichen oder Minuszeichen zu drucken.

```
PRINT USING "##.##^^^^";234.56
2.35E+02

PRINT USING ".####^^^^-";-888888
.8889E+06

PRINT USING "+.##^^^^";123
+.12E+03
```

Wird ein Unterstreichungszeichen in der Formatzeichenfolge angegeben, so wird das nächste Zeichen als literales Zeichen ausgegeben.

```
PRINT USING"  !##.## !";12.34
!12.34!
```

Ein Unterstreichungszeichen am Anfang der Formatzeichenfolge kann weggelassen werden.

```
PRINT USING "Ihrer Datei wurde ## zugewiesen";2

Ihrer Datei wurde #2 zugewiesen.
```

Das literale Zeichen selbst kann ein Unterstreichungszeichen sein, wenn Sie " " in die Formatzeichenfolge setzen.

% (Prozentzeichen)
Ist die auszudruckende Zahl größer als das angegebene numerische Feld, so wird vor der Zahl ein Prozentzeichen ausgedruckt. Überschreitet die Zahl beim Abrunden das Feld, so wird vor der abgerundeten Zahl ein Prozentzeichen ausgedruckt.

```
PRINT USING "##.##";111.22
%111.22

PRINT USING ".##";.999
%1.00
```

Überschreitet die Anzahl von angegebenen Ziffern 24, so kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf).

PRINT# und PRINT# USING-Befehle

Syntax: PRINT#<Dateinummer>,[USING <Zeichenfolgenausdruck>;]<Liste mit Ausdrücken>

Verwendung: Schreibt Daten in eine sequentielle Datei.

Bemerkungen/ Beispiele: <Dateinummer> ist die Nummer, die beim Eröffnen der Datei für die Ausgabe benutzt wurde. <Zeichenfolgenausdruck> besteht aus Formatierzeichen, wie in PRINT USING beschrieben. Die Ausdrücke in <Liste mit Ausdrücken> sind die numerischen und/oder Zeichenfolgenausdrücke, die in die Datei geschrieben werden.

PRINT# komprimiert keine Daten. Ein Abbild der Daten wird in die Datei geschrieben, genau so wie es mit PRINT auf dem Bildschirm angezeigt würde. Aus diesem Grund muß darauf geachtet werden, die Daten abzugrenzen, damit sie später aus der Datei richtig eingegeben werden.

In der <Liste mit Ausdrücken> werden die numerischen Ausdrücke durch Semikolon voneinander getrennt. Zum Beispiel:

```
PRINT#1,A;B;C;X;Y;Z
```

(Werden Kommas als Abgrenzungszeichen benutzt, so werden die zusätzlichen Leerstellen, die zwischen die Druckfelder eingefügt werden, ebenfalls in die Datei geschrieben.)

Zeichenfolgenausdrücke müssen in der Liste durch Semikolon voneinander getrennt werden. Damit die Zeichenfolgenausdrücke richtig in der Datei formatiert werden, werden in der Liste mit Ausdrükken explizite Abgrenzungszeichen benutzt.

Zum Beispiel A$="CAMERA" und B$="93604-1". Mit dem Befehl

```
PRINT#1,A$;B$
```

würde CAMERA93604-1 in die Datei geschrieben. Da keine Abgrenzungszeichen vorhanden sind, kann diese Angabe nicht als zwei separate Zeichenfolgen eingegeben werden. Um dieses Problem zu umgehen, werden wie folgt explizite Abgrenzungszeichen in den PRINT#-Befehl eingefügt:

```
PRINT#1,A$;",";B$
```

Nun wird folgendes Abbild in die Datei geschrieben:

```
CAMERA,93604-1
```

Diese Angabe kann in Form von zwei Zeichenfolgenvariablen aus der Datei gelesen werden.

Enthalten die Zeichenfolgen selbst Kommas, Semikolons, signifikante führende Leerstellen, Zeilenschaltungen oder Zeilenvorschübe, so werden diese in expliziten Anführungszeichen in die Datei geschrieben, CHR$(34).

Zum Beispiel A$="CAMERA, AUTOMATISCH" und B$=" 93604-1". Mit dem Befehl

```
PRINT#1,A$;B$
```

würde folgendes Abbild in die Datei geschrieben:

```
CAMERA, AUTOMATISCH 93604-1
```

Der Befehl

```
INPUT#1,A$,B$
```

würde "CAMERA" in A$ und "AUTOMATISCH 93604-1" in B$ eingeben. Um diese beiden Zeichenfolgen richtig in der Datei voneinander zu trennen, werden mit CHR$(34) Anführungszeichen in das Dateiabbild geschrieben. Der Befehl

```
PRINT#1,CHR$(34);A$;CHR$(34);CHR$(34);B$
    ,CHR$(34)
```

schreibt folgendes Abbild in die Datei:

"CAMERA,AUTOMATISCH"" 93604-1"

Der Befehl

INPUT#1,A$,B$

würde "CAMERA, AUTOMATISCH" in A$ und "93604-1 in B$ eingeben.

PRINT# kann auch mit der USING-Option benutzt werden, um das Format der Datei zu steuern. Zum Beispiel:

PRINT#1,USING"$$###.##,";J;K;L

Hinweis: Siehe auch WRITE#, bei dem keine expliziten Abgrenzungszeichen erforderlich sind.

PSET-Befehl

siehe PRESET

PUT (Dateien)-Befehl

Syntax: PUT[#]<Dateinummer>[,<Satznummer>]

Verwendung: Schreibt einen Datensatz aus einem Direktzugriffspuffer in eine Direktzugriffsdatei.

Bemerkungen: <Dateinummer> ist die Nummer, unter der die Datei eröffnet wurde. Wird "Satznummer" weggelassen, so wird dem Satz die nächste verfügbare Satznummer (nach dem letzten PUT) zugewiesen. Die größtmögliche Satznummer ist 32,767. Die kleinste Satznummer lautet 1.

Mit den Befehlen LSET, RSET, PRINT#, PRINT# USING und WRITE# können Zeichen in den Direktzugriffspuffer gesetzt werden, bevor ein PUT-Befehl ausgeführt wird. Bei WRITE# füllt GW-BASIC den Puffer bis zur Zeilenschaltung mit Leerzeichen auf.

Jeder Versuch, über das Ende des Puffers hinauszuschreiben, führt zu einer Fehlermeldung "Field overflow" (Feldüberlauf).

PUT kann auch für eine Datenübertragungsdatei benutzt werden. In diesem Fall ist "Satznummer" keine Satznummer, sondern stellt die Anzahl von Bytes für die Ausgabe dar. Hier muß darauf geachtet werden, daß diese Zahl nicht größer ist, als die mit der Option LEN in OPEN COM festgelegte Zahl.

Hinweis: Ein einzelner PUT-Befehl bedeutet nicht unbedingt, daß das Diskettenlaufwerk des Computers sofort aktiviert wird. GW-BASIC und das Betriebssystem versuchen nämlich, eine Reihe von Sätzen zu sammeln, bevor sie auf Diskette geschrieben werden.

PUT (Grafik)-Befehl

Syntax: PUT (x,y),<Matrix>[,<Abbild>]

Verwendung: Legt die Farben von Punkten auf dem Bildschirm mit Hilfe von Daten fest, die in einer Matrix gespeichert sind.

Bemerkungen: "x,y" sind die Koordinaten der oberen linken Position des Bildschirmbereichs, für den dieser Befehl gilt.

<Matrix> ist der Name der numerischen Matrix, die die Daten enthält. In der Beschreibung des GET (Grafik)-Befehls wird erläutert, wie grafische Daten in einer derartige Matrix gespeichert werden.

Wird <Abbild> angegeben, so kann zwischen verschiedenen Effekten ausgewählt werden.

Mit PSET werden die Grafikdaten so auf den Bildschirm gesetzt, wie sie mit GET in die Matrix gespeichert wurden.

PRESET hat dieselbe Auswirkung wie PSET, allerdings wird ein negatives Abbild erzeugt: bei einem Wert 0 in der Matrix wird Farbe 3 aus der aktuellen Farbpalette für den Bildschirmpunkt ausgewählt und umgekehrt; bei einem Wert 1 wird Farbe 2 für den Bildschirmpunkt ausgewählt und umgekehrt.

AND bewirkt, daß nur die Punkte der Matrix, die schon in einer Nicht-Hintergrundfarbe auf dem Bildschirm dargestellt werden, angezeigt werden. Die restlichen Teile eines schon auf dem Bildschirm stehenden Abbildes werden gelöscht.

Durch OR wird das Abbild der Matrix über das bestehende Bildschirmabbild gelegt.

XOR ist besonders bei Trickbildern von Nutzen. Hat ein Punkt auf dem Bildschirm dieselbe Farbe wie der entsprechende Punkt in der Matrix, so wird ein invertiert dargestelltes Abbild erzeugt. Wird für ein Abbild zwei Mal PUT mit XOR in derselben Position benutzt, so wird der ehemalige Hinter-

grund wieder hergestellt. Mit dieser Eigenschaft kann ein Objekt über den Bildschirm bewegt werden, ohne den Hintergrund zu beeinflussen:

1. Mit XOR wird das Abbild auf den Bildschirm gesetzt (PUT).

2. Die nächste Position für das Abbild wird berechnet.

3. Nun wird das Abbild erneut mit XOR in der bei 1 benutzten Position auf den Bildschirm gesetzt.

4. Nun wird mit der neuen Position des Abbilds zu Schritt 1 gegangen.

Das Standard-<Abbild> ist XOR.

In den folgenden Tabellen werden die Ergebnisse der Festlegung eines Bildschirmpunktes (mit PUT) für AND, OR und XOR bei Farbgrafiken mit niedriger und hoher Auflösung dargestellt. Farbe 0 ist die Hintergrundfarbe; die Farben 1, 2 und 3 sind die Farben der aktuellen Farbpalette.

AND		Matrixwert für Punkt			
		0	1	2	3
Alte Farbe des Bildschirm-punktes	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

OR		Matrixwert für Punkt			
		0	1	2	3
Alte Farbe des Bildschirm-punktes	0	0	1	2	3
	1	1	1	3	3
	2	2	3	2	3
	3	3	3	3	3

XOR		Matrixwert für Punkt			
		0	1	2	3
Alte Farbe des Bildschirm-punktes	0	0	1	2	3
	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0

Beispiel:

Bei dem folgenden Programm wird das XOR-"Abbild"
des PUT-Befehls benutzt, während ein grüner Ball über den numerischen Tastenblock der Tastatur auf dem Bildschirm hin- und herbewegt wird. (Zur Aktivierung der numerischen Funktion dieser Tasten anstelle der Funktion zur Bewegung des Cursors wird die Taste Num Lock betätigt.)

Mit Zeile 50 wird ein roter Kreis gezeichnet. Mit Zeile 55 wird dieser Kreis grün ausgefüllt. Da der Durchmesser des Kreises 20 Bildschirmpunkte beträgt, wird ein Bereich von 20 x 20 Bildschirmpunkten von dem GET-Befehl (Zeile 70) gespeichert, nachdem die obere linke Ecke dieses Bereichs er-mittelt wurde (Zeile 60). Mit Zeile 150 wird der grüne Ball wieder auf dem Bildschirm angezeigt. Zeile 160 ruft eine Subroutine auf, die X% und Y% je nach der betätigten Zifferntasten verschobene Werte zuweisen. ("7" führt zu einem Minus in der x- und y-Richtung; "9" führt zu einem Plus in der x-Richtung und einem Minus in der y-Richtung usw). In Zeile 170 wird das Abbild des Balls wieder mit XOR in die alte Position gesetzt. Auf diese Weise wird das Abbild gelöscht und der Ball sofort in der nächsten Position angezeigt, wobei die in X% und Y% enthaltenen verschobenen Werte benutzt werden.

Mit der Fehlerunterbrechungsroutine wird die Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) ausgegeben, wenn versucht wird, ein Abbild in einen Bereich außerhalb des Bildschirms zu setzen.

```
  5 ON ERROR GOTO 1100
 10 DIM BALL%(64)
 20 PI=3.141593
 30 SCREEN 1:CLS
 40 COLOR 0,0
 50 CIRCLE (160,100),10,2
 55 PAINT STEP (0,0),1,2
 60 PRESET STEP (-10,-10)
 70 GET
    (POINT(0),POINT(1))-STEP(20,20),BALL%
100 CLS
150 PUT STEP(X%,Y%),BALL%,XOR
160 GOSUB 1000
170 PUT STEP(0,0),BALL%,XOR
180 GOTO 150
500 REM
1000 REM ******** Betrachten Sie die Tastatur
1001 IF INKEY$="" THEN GOTO 1001
```

```
1005 X%=0:X%=0
1010 FOR LOOL%=1 TO 5
1020 K$=INKEY$
1030 X%=X%-4*(K$="9" OR K$="6" OR
     K$="3") + 4*(K$="7" OR K$="4" OR
     K$="1")
1040 Y%=4*(K$="3" OR K$="2" OR
     K$="1") + 4*(K$="9" OR K$="8" OR
     K$="7")
1050 NEXT LOOK%
1060 RETURN
1100 REM ******** Unterbrechung, wenn Ball
     vom Bildschirm läuft
1110 IF ERR=5 THEN BEEP:RESUME 50
1120 ERROR GOTO 0
```

Ein Flackern des Bildschirms wird auf ein Mindestmaß beschränkt, indem das neue Abbild unmittelbar nach dem Löschen des alten Abbildes angezeigt wird (Zeile 180).

Hinweis:

Der Hintergrund muß nicht unbedingt bewahrt werden. Sie können auch eine einzelne Matrix, die sowohl das neue Abbild als auch ausreichend Hintergrundpunkte für das alte Abbild enthält, mit PUT auf den Bildschirm setzen, wobei PSET als "Abbild" benutzt wird. Auf diese Weise wird einer der beiden PUT-Befehle beibehalten, die bei Benutzung von XOR "Abbild" erforderlich sind. Dies bedeutet jedoch auch, daß die Matrix entsprechend größer sein muß.

RANDOMIZE-Befehl

Syntax: RANDOMIZE [<numerischer Ausdruck>]
oder RANDOMIZE TIMER

Verwendung: Setzt einen neuen Ausgangspunkt für den Zufallszahlengenerator.

Bemerkungen: Wird <numerischer Ausdruck> weggelassen, so stellt GW-BASIC die Programmausführung ein und fordert durch Ausgabe von:

Random Number Seed (-32768 to 32767)?
(Ausgangszufallszahl (-32769 bis 32767)?

einen Wert an, bevor RANDOMIZE ausgeführt wird.

Wird kein neuer Ausgangspunkt für den Zufallszahlengenerator gesetzt, so gibt die RND-Funktion bei jeder Ausführung des Programms dieselbe Folge von Zufallszahlen zurück. Um die Änderung der Folge von Zufallszahlen bei jeder Ausführung des Programms zu ändern, wird ein RANDOMIZE-Befehl an den Anfang des Programms gesetzt und der "numerische Ausdruck" bei jeder Ausführung geändert.

Beispiel:

```
10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RDN;
40 NEXT I
```

ergibt

Ausgangszufallszahl (-32768 to 32767)?

Geben Sie 3 als Antwort ein. Nun werden fünf Zufallszahlen angezeigt. Danach wird das Programm erneut gestartet und eine andere Zahl im zulässigen Bereich eingegeben. GW-BASIC zeigt eine andere Gruppe von Zahlen an.

Wird das Programm erneut mit Eingabe von 3 gestartet, so werden Sie feststellen, daß dieselben Zahlen wie bei der ersten Ausführung des Programms angezeigt werden. Die Zahlen sehen nicht wie reine Zufallszahlen aus (siehe RND).

Hinweis:

Häufig ist es unpraktisch, wenn ein Bedienereingriff erforderlich ist, um einen neuen Ausgangspunkt für den Zufallszahlengenerator zu bestimmen.

Aus diesem Grund kann mit der TIME$-Funktion der "numerische Ausdruck" geliefert werden. In diesem Fall ist es am sinnvollsten, den Zähler für die Sekunden zu lesen:

```
RANDOMIZE VAL (RIGHT$(TIM$,2))
```

Mit GW-BASIC 2.0 (siehe "Systemkompatibilität" in diesem Kapitel) kann die TIMER-Funktion benutzt werden. Bei dieser Funktion wird die VAL-Transformation umgangen. Außerdem ermöglicht sie einen breiteren Wertebereich, mit dem ein neuer Ausgangspunkt für den Zufallszahlengenerator festgelegt werden kann:

```
RANDOMIZE TIMER MOD 32767
```

READ-Befehl

Syntax: READ <Liste mit Variablen>

Verwendung: Liest Datenelemente aus einer Datenliste und weist sie Variablen zu.

Bemerkungen: Ein READ-Befehl muß stets in Verbindung mit DATA benutzt werden. READ weist Variablen ein Datenelement nach dem anderen zu. Ein aus einer DATA-Liste gelesenes Datenelement muß denselben Typ aufweisen wie die Variable, der es zugewiesen wird. Stimmen die Typen nicht überein, so wird eine Fehlermeldung "Syntax error" (Syntaxfehler) angezeigt.

Ein einzelner READ-Befehl kann auf ein oder mehrere DATA-Elemente zugreifen (der Zugriff erfolgt in der Reihenfolge, in der die Datenelemente angegeben werden). Mehrere READ-Befehle können auf dieselben DATA-Elemente zugreifen. Überschreitet die Anzahl von Variablen in "Liste mit Variablen" die Anzahl von Datenelementen, so wird eine Fehlermeldung "Out of data" (Keine Daten mehr) angezeigt. Ist die angegebene Anzahl von Variablen kleiner als die Anzahl von Datenelementen, so beginnen nachfolgende READ-Befehle mit dem Lesen der Daten bei dem ersten ungelesenen Datenelement. Sind keine nachfolgenden READ-Befehle vorhanden, so werden die zusätzlichen Daten ignoriert.

Sollen die Daten von Anfang an neu gelesen werden, so wird der RESTORE-Befehl benutzt.

Beispiele:

```
.
.
.
 80 FOR I=1 TO 10
 90 READ A(I)
100 NEXT I
110 DATA 3.08,5.19,3.12,3.98,4.23
120 DATA 5.08,5.55,4.00,3.16,3.37
```

.
.

Dieses Programmsegment liest die Werte aus den Datenlisten (mit READ) in die (implizit definierte) Matrix A.

Nach der Ausführung lautet der Wert von A(1) 3.08 usw.

```
10 PRINT "PLZ", "STADT", "STADTBEZIRK"
20 READ C$,S$,Z
30 DATA "8000 MÜNCHEN 40"
40 PRINT C$,S$,Z
ergibt
PLZ  STADT       STADTBEZIRK
8000 MÜNCHEN 40
```

Dieses Programm liest Zeichenfolgen und numerische Daten aus den Datenelementen in Zeile 30. Beachten Sie, daß Datenelemente mit Kommas, Semikolons oder signifikanten führenden bzw. abschließenden Leerstellen in Anführungszeichen stehen müssen.

REM-Befehl

Syntax: REM <Bemerkung>

Verwendung: Mit diesem Befehl können erläuternde Bemerkungen in ein Programm aufgenommen werden.

Bemerkungen: REM-Zeilen werden nicht ausgeführt, sondern genau so ausgegeben, wie sie bei Auflistung des Programms eingegeben wurden.

In REM-Zeilen kann von einem GOTO- oder GOSUB-Befehl gegangen werden. Die Ausführung wird mit dem ersten ausführbaren Befehl im Anschluß an die REM-Zeile fortgesetzt.

Bemerkungen können an das Ende einer Zeile angefügt werden, indem vor die Bemerkung ein Apostroph anstelle von :REM gesetzt wird.

Beispiel:

```
.
.
.
120 REM DURCHSCHNITTSGESCHWINDIG-
    KEIT BERECHNEN
130 FOR I=1 TO 20
140 SUM=SUM + V(I)
.
.
.
oder
.
.
.
120 FOR I=1 TO 20 'DURCHSCHNITTS-
    GESCHWINDIGKEIT BERECHNEN
130 SUM=SUM+V(I)
140 NEXT I
.
.
.
```

Hinweis: Enthält eine Programmzeile mehr als einen Befehl, so muß REM – sofern vorhanden – der letzte Befehl in dieser Zeile sein.

Mit REM-Zeilen oder angefügten Bemerkungen, vor denen ein Apostroph steht, kann eine Programmauflistung in Abschnitte unterteilt und erläutert werden, wie das Programm arbeitet. Das Programm sollte jedoch nicht mit zu vielen Bemerkungen versehen werden, da sie Speicherplatz erfordern und die Ausführungszeit verlängern.

RENUM-Befehl

Syntax: RENUM [<neue Nummer>][,[<alte Nummer>] [,<Erhöhung>]]

Verwendung: Numeriert Programmzeilen neu.

Bemerkungen: <Neue Nummer> ist die erste Zeilennummer, die in der neuen Folge benutzt werden soll. Der Standardwert lautet 10. <Alte Nummer> entspricht der Zeile in dem aktuellen Programm, bei der mit der Neunumerierung begonnen werden soll. Standardmäßig handelt es sich hier um die erste Zeile des Programms. "Erhöhung" ist die in der neuen Folge zu benutzende Erhöhung. Der Standardwert lautet 10.

RENUM ändert auch sämtliche Bezugnahmen auf Zeilennummern im Anschluß an GOTO, GOSUB, THEN, ELSE, ON...GOTO, ON...GOSUB, RESTORE und RESUME Befehle, um die neuen Zeilennummern wiederzugeben.

Hinweis: Mit RENUM kann die Reihenfolge der Programmzeilen nicht geändert werden (z.B. RENUM 15,30, wenn das Programm über drei Zeilen mit den Nummern 10, 20 und 30 verfügt). Auch können mit diesem Befehl keine Zeilennummern erstellt werden, die größer sind als 65529. Ansonsten wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt.

Beispiele: RENUM

Numeriert das gesamte Programm neu. Die erste neue Zeilennummer ist 10. Die Zeilennummern werden um jeweils 10 erhöht.

RENUM 300,,50

Numeriert das gesamte Programm neu. Die erste Zeilennummer ist 300. Die Zeilen werden um jeweils 50 erhöht.

RENUM1000,900,20

Numeriert die Zeilen ab 900 in aufsteigender Reihenfolge neu, so daß mit Zeile Nr. 1000 begonnen wird. Die Zeilennummern werden um jeweils 20 erhöht.

RESET-Befehl

Syntax: RESET

Verwendung: Schließt sämtliche Dateien in sämtlichen Laufwerken ab.

Bemerkungen: RESET schließt sämtliche eröffneten Dateien in allen Laufwerken ab und ergänzt das Inhaltsverzeichnis für jede Diskette, die über eröffnete Dateien verfügte.

Sämtliche Dateien müssen abgeschlossen werden, bevor eine Diskette aus dem Laufwerk herausgenommen werden kann.

Hinweis: Für das Abschließen einzelner Dateien wird CLOSE benutzt.

RESTORE-Befehl

Syntax: RESTORE [<Zeilennummer>]

Verwendung: Mit diesem Befehl können DATA-Befehle ab einer angegebenen Zeile neu gelesen werden.

Bemerkungen: Nach Ausführung eines RESTORE-Befehls greift der nächste READ-Befehl auf das erste Element in der ersten DATA-Liste in dem Programm zu. Wird <Zeilennummer> angegeben, so greift der nächste READ-Befehl auf das erste Datenelement in der angegebenen DATA-Zeile zu.

Beispiel:

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,79
50 PRINT A,B,C;D;E;F
ergibt
57 68 79 57 68 79
```

RESUME-Befehl

Syntax:

RESUME

RESUME 0

RESUME NEXT

RESUME <Zeilennummer>

Verwendung: Setzt die Programmausführung fort, nachdem eine fehlerbehebende Maßnahme ausgeführt wurde.

Bemerkungen: Jede der oben angeführten vier Syntaxformen kann benutzt werden, je nachdem, an welcher Stelle die Ausführung wieder aufgenommen werden soll:

RESUME oder RESUME 0

Die Ausführung wird bei dem Befehl wieder aufgenommen, der den Fehler verursacht hat.

RESUME NEXT

Die Ausführung wird bei dem Befehl wieder aufgenommen, der unmittelbar auf den fehlerverursachenden Befehl folgt.

RESUME <Zeilennummer>

Die Ausführung wird bei <Zeilennummer> wieder aufgenommen.

Ein RESUME-Befehl, der nicht in einer Fehlerbearbeitungs-Routine steht, führt zur Anzeige der Fehlermeldung "RESUME without error" (RESUME ohne Fehler).

Beispiel:

```
10 ON ERROR GOTO 900
 .
 .
 .
900 IF (ERR=230)AND(ERL=90) THEN PRINT
    "ERNEUT VERSUCHEN":RESUME 80
```

.
.
.

Hinweis: RENUMBER versucht nicht, die Zahl 0 in einem RESUME 0 Befehl zu ändern. Um Sie darauf hinzuweisen, gibt GW-BASIC eine Fehlermeldung "Undefined line number" (Undefinierte Zeilennummer) aus.

RETURN-Befehl

Syntax: RETURN [<Zeilennummer>]

<Zeilennummer> gibt die Nummer der Programmzeile an, zu der GW-BASIC nach Ausführung einer Subroutine zurückkehrt.

Verwendung: Ermöglicht den Rücksprung aus einer Subroutine, in die mit GOSUB oder ON GOSUB gegangen wurde, in das Hauptprogramm.

Bemerkungen: Die Benutzung von nicht-lokalen RETURN-Befehlen, d.h. die Benutzung von RETURN mit einer "Zeilennummer" setzt besondere Vorsicht voraus. Sie müssen darauf achten, daß ein derartiger RETURN-Befehl nicht den RETURN-Befehl für eine andere noch aktive Subroutine umgeht.

RIGHT$-Funktion

Syntax: RIGHT$(<Zeichenfolgenausdruck>,<Anzahl>)

Verwendung: Gibt die rechtsbündigste <Anzahl> von Zeichen des <Zeichenfolgenausdrucks> zurück.

Bemerkungen: Ist <Anzahl> größer als oder gleich der Anzahl von Zeichen in dem <Zeichenfolgenausdruck>, so wird die gesamte Zeichenfolge zurückgegeben. Ist <Anzahl> gleich 0, so wird die Nullzeichenfolge (Zeichenfolge mit der Länge 0) zurückgegeben.

Beispiel:

```
10 A$="DISK BASIC"
20 PRINT RIGHT$(A$,5)
ergibt
BASIC
```

Siehe auch die LEFT$- und MID$-Funktionen.

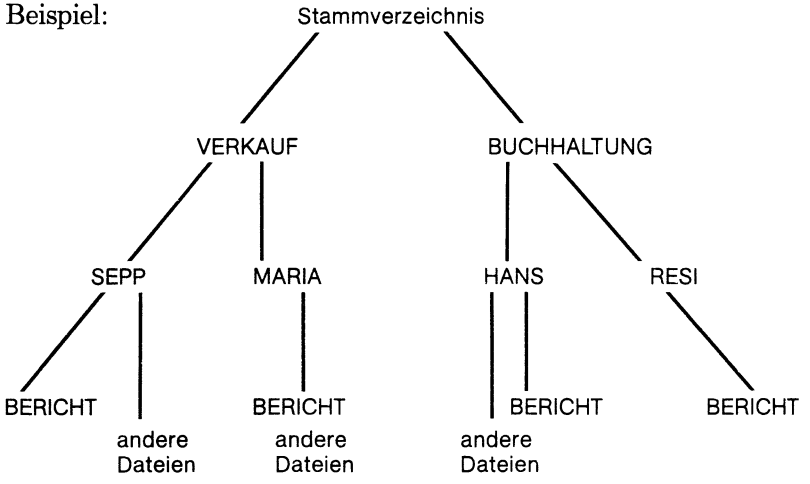
RMDIR-Befehl

Syntax: <Pfad>

Verwendung: Löscht ein Inhaltsverzeichnis aus dem angegebenen Laufwerk.

Bemerkungen: <Pfad> ist ein Zeichenfolgenausdruck, der 128 Zeichen nicht überschreitet und ein Unterverzeichnis kennzeichnet, das gelöscht werden soll. Für Einzelheiten über Pfade und Inhaltsverzeichnisse wird auf das NCR-DOS Handbuch verwiesen.

Beispiel:



Wird von der oben angegebenen hierarchischen Struktur ausgegangen und befinden Sie sich gerade in dem Stammverzeichnis, so löscht der Befehl

RMDIR "VERKÄUFE \ BUCHHALTUNG \ SUSANNE"

das Inhaltsverzeichnis SUSANNE auf dem angegebenen Pfad, vorausgesetzt, unter SUSANNE sind keine Dateien und Unterverzeichnisse vorhanden sind. In diesem Beispiel müßte zuerst die Datei BERICHT mit KILL gelöscht werden, bevor der MRDIR-Befehl ausgegeben werden kann.

Hinweis: Bei GW-BASIC kann das übergeordnete Inhaltsverzeichnis des Inhaltsverzeichnisses nicht

gelöscht werden, mit dem gerade gearbeitet wird. Wird dies dennoch versucht, so wird eine Fehlermeldung "Path/file access" (Pfad-/Datei-Zugriff) angezeigt. Derselbe Fehler wird angezeigt, wenn noch Unterverzeichnisse des zu löschenden Inhaltsverzeichnisses vorhanden sind, oder wenn in dem Inhaltsverzeichnis noch Dateien vorhanden sind.

Außerdem wird eine Fehlermeldung "File not found" (Datei nicht gefunden) angezeigt, wenn Sie versuchen, ein Inhaltsverzeichnis mit KILL zu löschen.

RND-Funktion

Syntax: RND[(X)]

Verwendung: Gibt eine Zufallszahl mit einfacher Genauigkeit zwischen 0 und 1 zurück.

Bemerkungen: Die RND-Funktion bezieht sich auf eine Folge von Pseudo-Zufallszahlen, die intern von GW-BASIC gespeichert wurden. Diese Folge ist so aufgebaut, daß sie in jeder Beziehung eine Folge von Zufallszahlen darstellt.

Dieselbe Folge von Zufallszahlen wird bei jeder Ausführung des Programms generiert, es sei denn, es wird ein anderer Ausgangspunkt für den Zufallszahlengenerator festgelegt (siehe RANDOMIZE).

Ist X größer als 0 oder wird X weggelassen, so wird die nächste Zufallszahl in der Folge generiert. Ist X gleich 0, so wird die letzte generierte Zahl wiederholt.

Bei einem negativen Wert von X wird ein neuer Ausgangspunkt für den Zufallszahlengenerator festgelegt, wobei sich jedoch RANDOMIZE nicht auswirkt.

RND erreicht niemals wirklich den Wert 1. Soll deshalb ein ganzzahliger Zufallswert in dem Bereich von 0 bis 10 einschließlich zurückgegeben werden, so wird beispielsweise folgender Befehl ausgegeben:

```
PRINT INT(RND*11))
```

(Dies ist nicht unbedingt erforderlich, wenn der Befehl bzw. die Funktion, die RND aufruft, eine Abrundung auf die nächste Ganzzahl vornimmt.)

Beispiele: Das folgende Beispiel vermittelt eine Vorstellung von einer Folge mit Pseudo-Zufallszahlen. (Dieses Programm muß mit <Ctrl-Break> verlassen werden.)

```
10 SCREEN 2:CLS
```

```
20 PSET (RND*639,RND*199)
30 GOTO 20
```

Nun soll dasselbe Programm erneut ausgeführt werden, dieses Mal jedoch mit größeren Möglichkeiten:

```
10 SCREEN 2:CLS
20 RANDOMIZE TIMER MOD 32767
30 PSET (RND*639,RND*199)
40 GOTO 20
```

In dem folgenden Programm werden wiederholt zwei Würfel geworfen, die beiden Zahlen addiert und je nach Ergebnis zu Spalte 2 bis 12 hinzugefügt:

```
 10 DEFINIT A-Z
 20 DIM STAT(12)
 30 FOOT=180
 40 SCREEN 1:COLOR 0,0:CLS
 50 PRESET (93,14)
 60 DRAW "c2 r28 d28 128 u28 br56"
 70 DRAW "r28 d28 128 u28"
 80 LOCATE 25,2:PRINT "2 3 4 5 6 7 8 9 10 11 12"
 90 FOR DICE=1 TO 1000
100 RANDOMIZE TIMER MOD 32767
110 D1=INT(RND*6)+1:D2=INT(RND*6)+1
120 LOCATE 3,13:PRINT D1
130 LOCATE 3,20:PRINT D2
140 THROW=D1+D2
150 LOCATE 3,30
160 PRINT"-->";THROW
170 STAT(THROW)=STAT(THROW)+1
180 PSET
    (24*THROW-36,FOOT-STAT(THROW)),1
190 FOR D=1 TO 800:NEXT D
200 NEXT DICE%
```

RSET-Befehl

siehe LSET

RUN-Befehl

Syntax 1: RUN [<Zeilennummer>]

Verwendung: Führt ein gerade im Speicher stehendes Programm aus.

Bemerkungen: Wird <Zeilennummer> angegeben, so beginnt die Ausführung mit dieser Zeile. Ansonsten beginnt die Ausführung mit der niedrigsten Zeilennummer.

Syntax 2: RUN <Dateispez> [,R]

Verwendung: Lädt eine Datei von Diskette in den Speicher und führt sie aus.

Bemerkungen: Die "Dateispez" muß den Dateinamen enthalten, der beim Sichern der Datei benutzt wurde. (GW-BASIC fügt die Dateinamenerweiterung .BAS hinzu, wenn Sie keine Dateinamenerweiterung angeben.)

RUN schließt sämtliche eröffneten Dateien ab und löscht den aktuellen Speicherinhalt, bevor das angegebene Programm geladen wird. Wird jedoch die Option "R" angegeben, so bleiben sämtliche Datendateien geöffnet.

SAVE-Befehl

Syntax: SAVE <Dateispez> [{,A}{,p}]

Verwendung: Sichert eine Programmdatei auf Diskette.

Bemerkungen: "Dateispez" ist ein Zeichenfolgenausdruck, entsprechend den NCR-DOS-Regeln für das Benennen von Dateien angegeben wird. GW-BASIC fügt die Standard-Dateinamenerweiterung .BAS hinzu, wenn mit dem SAVE-Befehl keine Dateinamenerweiterung angegeben wird. Ist ein Dateiname schon vorhanden, so wird die Datei überschrieben.

Wird die Option A angegeben, so wird die Datei im ASCII-Format gesichert. Wird die Option A nicht angegeben, so sichert GW-BASIC die Datei in einem komprimierten Binär-Format. Das ASCII-Format belegt mehr Platz auf der Diskette. Bei einigen Zugriffsarten ist es jedoch erforderlich, daß die Dateien das ASCII-Format aufweisen. So setzt der MERGE-Befehl beispielsweise eine Datei im ASCII-Format voraus. Eine Datei im ASCII-Format kann mit einem Editor oder mit dem NCR-DOS Befehl TYPE angezeigt werden.

Mit der Option P wird die Datei geschützt, indem sie in einem codierten Binärformat gesichert wird. Wird eine geschützte Datei später mit RUN ausgeführt (oder mit LOAD geladen), so führt jeder Versuch, die Datei aufzulisten oder zu editieren zu der Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf). Diese Eigenschaft kann nicht umgangen werden, so daß stets eine ungeschützte Version des Programms aufbewahrt werden sollte, die dann später aufgelistet oder editiert werden kann.

Beispiele: SAVE "B:COM2",A

Sichert das Programm COM2 im ASCII-Format auf der Diskette in Laufwerk B.

SAVE "ENIGMA",P

Sichert das Programm ENIGMA als geschützte Datei, die nicht geändert werden kann.

SCREEN-Befehl

Syntax: SCREEN [<Modus>][,<aktiv>][,<Ausgabeseite>][,<Anzeigeseite>]

Verwendung: Legt die Bildschirm-Attribute für einen Farbbildschirm fest.

Bemerkungen: <Modus> ist ein ganzzahliger Ausdruck 0, 1, 2, 3 oder 4, der den Anzeigemodus festlegt: Textmodus, Grafik mit niedriger Auflösung, Grafik mit mittlerer Auflösung, einfarbige Grafik mit hoher Auflösung, bzw. Farbgrafik mit hoher Auflösung.

<Aktiv> ist ein ganzzahliger Ausdruck 0 oder 1. Dieses Attribut ist mit der Aktivierung und Deaktivierung der Farbe in früheren Versionen von BASIC verknüpft. Es hat keine Auswirkungen auf die Bildschirmanzeige bei dem NCR-PC, wenn die gelieferte GW-BASIC Version benutzt wird.

<Ausgabeseite> bezieht sich auf die Nummer der Seite im Zeichenmodus, in die GW-BASIC die Bildschirmausgabe schreiben soll. Zulässige Werte sind die Werte 0 bis 7 bei einer Zeilenbreite von 40 Zeichen bzw. 0 bis 3 bei einer Zeilenbreite von 80 Zeichen.

<Anzeigeseite> bezieht sich auf die Nummer der Seite im Zeichenmodus, die GW-BASIC anzeigt. Wird hier kein Wert angegeben, so sind "Aus-gabeseite" und "Anzeigeseite" identisch.

Bei Ausführung eines gültigen SCREEN-Befehls setzt GW-BASIC die Vordergrundfarbe auf weiß und die Hintergrundfarbe auf schwarz. (Die Farben können dann später mit dem COLOR-Befehl geändert werden.)

Unterscheidet sich der "Modus" von dem vorher aktiven Modus, so wird der Bildschirm gelöscht. Wird bei Festlegung von "Modus" ein gelöschter Bildschirm benötigt, so wird empfohlen, einen nachfolgenden CLS-Befehl aufzunehmen, da er unabhängig von dem vorhergehenden Modus benutzt werden kann.

Werden unterschiedliche Seitennummern für "Anzeigeseite" und "Ausgabeseite" angegeben, so kann

das Programm in "Ausgabeseite" schreiben, ohne die aktuelle Bildschirmanzeige zu beeinträchtigen. Mit einem nachfolgenden SCREEN-Befehl kann der "Anzeigeseite" dann dieselbe Nummer wie der "Ausgabeseite" zugewiesen werden.

In diesem Fall wird die im Hintergrund erstellte Bildschirmseite sofort angezeigt. Für sämtliche Seiten gibt es nur einen Cursor. Deshalb wird POS und CSRLIN benutzt, wenn der Cursor später in eine bestimmte Position auf einer Seite bewegt werden soll, bevor eine weitere Seite zur <Anzeigeseite> gemacht wird. Mit LOCATE wird der Cursor dann in die auf diese Weise gespeicherte Position bewegt.

Jeder beliebige Parameter kann weggelassen werden, indem an seiner Stelle ein Komma benutzt wird. Der alte Wert für diesen Parameter wird dann beibehalten. Dies gilt jedoch nicht für "Anzeigeseite", die standardmäßig auf "Ausgabeseite" gesetzt wird.

Beispiele:

10 SCREEN 0,1,0,0

Wählt den Zeichenmodus aus und setzt "Ausgabeseite" und "Anzeigeseite" auf 0.

20 SCREEN ,,1,0

Beläßt den Anzeige-"Modus" unverändert und setzt "Ausgabeseite" und "Anzeigeseite" auf 1 bzw. 0.

10 SCREEN 1:CLS

Schaltet in Grafiken mit niedriger Auflösung um oder bestätigt diese und löscht den Bildschirm mindestens ein Mal.

10 SCREEN 3

Schaltet in einfarbige Grafik mit hoher Auflösung um oder bestätigt diese.

Hinweis:

Bezüglich der Größe der Zeichen, die im Grafikmodus angezeigt werden, wird auf WIDTH verwiesen.

Soll das Programm sowohl auf einem Monochrom als auch auf einem Farb-Bildschirm ausgeführt werden, so geben Sie

SCREEN 0

an.

SCREEN-Funktion

Syntax: SCREEN(<Zeile>,<Spalte>[,<Attribut>])

Verwendung: Gibt den ASCII-Code (0 bis 255) des gerade in einer bestimmten Position auf dem Bildschirm angezeigten Zeichens zurück.

Bemerkungen: <Zeile> ist ein numerischer Ausdruck in dem Bereich von 1 bis 25, mit dem die Zeilennummer angegeben wird.

<Spalte> ist ein numerischer Ausdruck in dem Bereich von 1 bis 40 oder von 1 bis 80 (je nach WIDTH), mit dem die Spaltennummer angegeben wird.

<Attribut> ist nur im Zeichenmodus zulässig. Handelt es sich bei "Attribut" um einen von 0 abweichenden Wert, so wird eine Zahl zurückgegeben, die die Anzeigeeigenschaften der angegebenen Zeichenposition darstellt. Diese Zahl liegt in dem Bereich von 0 bis 255:

- Der Rest aus der Division dieser Zahl durch 16 (Zahl MOD 16) gibt den Code der Zeichenfarbe an (siehe COLOR).
- Die Hintergrundfarbe wird folgendermaßen berechnet

  ((Zahl-Schreiben)/16) MOD 129

  wobei Schreiben den Farbcode 0...15 des Zeichens darstellt.
- Ist die Zahl größer als 127, so blinkt das Zeichen auf.

Im Grafikmodus gibt die SCREEN-Funktion den ASCII-Code zurück, wenn bei der angegebenen Position ein ASCII-Zeichen angezeigt wird. Enthält die Position einen Teil der Grafik (Punkte, Linien, usw.), so ist der zurückgegebene Wert gleich 0.

Beispiele: 200 X% = SCREEN (10,1)

Weist X% den ASCII-Code des Zeichens in Zeile 10, Spalte 1 zu.

```
210 C% = SCREEN (10,1,1)
```

Weist C% eine Zahl in dem Bereich von 0 bis 255 zu, die die Anzeigeeigenschaften für diese Zeichenposition darstellt.

SGN-Funktion

Syntax: SGN(X)

Verwendung: Gibt den Wert von X relativ zu 0 an:

Ist X>0, so gibt SGN(X) 1 zurück.
Ist X=0, so gibt SGN(X) 0 zurück.
Ist X<0, so gibt SGN(X) -1 zurück.

Beispiel: ON SGN(X)+2 GOTO 100,200,300

Verzweigt sich zu 100, wenn X negativ ist, zu 200 wenn X gleich 0 ist und zu 300, wenn X positiv ist.

SHELL-Befehl

Syntax: SHELL [<Befehlsfolge>]

Verwendung: Lädt und führt eine NCR-DOS .EXE-, .COM- oder .BAT-Datei aus und kehrt danach zu dem GW-BASIC Programmbefehl zurück, der auf den SHELL-Befehl folgt.

Bemerkungen: Wird SHELL ohne eine "Befehlsfolge" angegeben, so wird die Systemeingabeaufforderung von NCR-DOS angezeigt, z.B. A>. Danach können die NCR-DOS .EXE-, .COM- oder .BAT-Dateien aufgerufen werden. Um zu GW-BASIC zurückzukehren, wird der NCR-DOS Befehl EXIT eingegeben. Dieser Befehl ist nur nötig, wenn der SHELL-Befehl ohne "Befehlsfolge" eingegeben wurde.

SHELL kann eine Befehlsfolge enthalten, die den NCR-DOS Regeln für die Ausgabe von Befehlen entpricht. Führen Sie Ihre eigene Befehlsdatei aus, so müssen Sie prüfen, ob diese nicht mit der Bedingung "Terminate and stay resident" (Beenden und resident bleiben) endet. Ansonsten gibt GW-BASIC die Fehlermeldung "Can't continue after SHELL" (Fortsetzung nach SHELL nicht möglich) aus. Ist nicht genügend Platz im Speicher vorhanden, um GW-BASIC gespeichert zu belassen und das Programm mit dem SHELL-Befehl auszuführen, so wird eine Fehlermeldung "Out of memory" (Kein Speicherplatz mehr) angezeigt.

Muß das Programm mit dem SHELL-Befehl eine Datei verarbeiten, so darf diese Datei zum Zeitpunkt der Ausführung des SHELL-Befehls nicht geöffnet sein. GW-BASIC kann die Datei erneut öffnen, sobald das Programm mit dem SHELL-Befehl beendet ist.

Beispiel:

```
10 OPEN "SORTIN.DAT" FOR OUTPUT AS #1
 .
 .
 .
1000 CLOSE #1
```

```
1010 SHELL"SORT<SORTIN.DAT
     >SORTOUT.DAT"
1020 OPEN "SORTOUT.DAT" FOR INPUT AS
     #1
  .
  .
  .
```

Bei diesem Beispiel wird der NCR-DOS Befehl SORT benutzt, um die während des GW-BASIC Programms eingegebenen Daten zu sortieren. Beachten Sie, daß die Datei abgeschlossen werden muß (Zeile 1000), bevor der SHELL-Befehl ausgeführt wird.

Hinweis:

Programmierer, die ihre eigenen NCR-DOS .COM- oder .EXE-Dateien schreiben, die durch GW-BASIC mit SHELL geladen und ausgeführt werden sollen, müssen darauf achten, daß der Unterbrechungsvektor sofort nach Eintritt in das Programm mit dem SHELL-Befehl gesichert und vor dem Rücksprung in GW-BASIC wiederhergestellt wird.

SIN-Funktion

Syntax: SIN(X)

Verwendung: Gibt den Sinus von X zurück, wobei X in Radianten angegeben wird.

Bemerkungen: Der Sinus wird mit einfacher Genauigkeit ausgewertet, es sei denn, beim Laden von GW-BASIC wird die Option /D angegeben.

Beispiel:
```
PRINT SIN(1.5)
ergibt
 .9984951
```

Hinweis: Sollen Radiant in Grad umgewandelt werden, so wird folgender Befehl ausgegeben:

```
DEGREES = RADIANS*180/PI
```

wobei PI (einfache Genauigkeit) 3.141593 beträgt.

Sollen Grad in Radiant umgewandelt werden, so wird folgender Befehl angegeben:

```
RADIANS = DEGREES*PI/180
```

SOUND-Befehl

Syntax: SOUND <Frequenz>,<Dauer>

<Frequenz>
Gibt die Frequenz in Hertz (Zyklen pro Sekunde) an. An dieser Stelle wird die gewünschte Zahl von 37 bis 32767 angegeben (siehe folgende Tabelle mit Noten und Frequenzen).

<Dauer>
Gibt die gewünschte Länge des Tons in Zeiteinheiten an. (Eine Zeiteinheit = 55 ms.) An dieser Stelle wird die gewünschte Anzahl von Zeiteinheiten in dem Bereich von 0 bis 65535 angegeben (siehe nachfolgende Tempo-Tabelle).

Verwendung: Generiert einen Ton über den Lautsprecher.

In der folgenden Tabelle werden die Noten mit den zugehörigen Frequenzen angegeben. Die Note A hat eine Frequenz von 440.

Note	Frequenz	Note	Frequenz
Pause	32767	F#	740
A	220	G	784
A#	233	G#	830
B	247	A	880
C	262	A#	930
C#	277.2	B	987.8
D	293.6	C	1046.4
D#	311.6	C#	1106
E	329.6	D	1174.6
F	349.2	D#	1244
F#	370	E	1318.6
G	392	F	1397
G#	416	F#	1480
A	440	G	1568
A#	466	G#	1660
B	493.2	A	1760
*C	523.2	A#	1864
C#	554.8	B	1975.6
D	587.4	C	2093
D#	622	C#	2217.4
E	659.2	D	2349.4
F	598.4		

*Mittleres C

Bemerkungen: SOUND erzeugt einen Ton, der ausgegeben wird, bis ein anderer SOUND-Befehl angetroffen wird. Wird ein SOUND-Befehl mit einer "Dauer" von 0 angetroffen, so wird jeder gerade ausgeführte Ton ausgeschaltet. (Wird gerade kein SOUND-Befehl ausgeführt, so hat SOUND "Frequenz",0 keine Auswirkung.)

Töne können gepuffert werden, damit die Programmausführung nicht gestoppt wird, wenn ein neuer SOUND-Befehl angetroffen wird. (Siehe MB-Befehl unter PLAY.)

Für Ruhepausen wird SOUND 32767, "Dauer" benutzt.

Die "Dauer" für einen Takt wird auf der Grundlage der Takte pro Minute berechnet. Die Takte pro Minute werden durch 1092 (die Anzahl von Zeiteinheiten in einer Minute) dividiert. In der folgenden Tabelle werden typische Tempi in der Form von Zeiteinheiten (Dauer) angegeben.

	Tempo	Takt-schläge/ Minute	Zeiteinheiten/ Takt (Dauer)
Sehr langsam	Larghissimo		
↓	Largo	40-60	27.3-18.2
↓	Larghetto	60-66	18.2-16.55
↓	Grave		
↓	Lento		
↓	Adagio	66-76	16.55-14.37
Langsam	Adagietto		
↓	Andante	76-108	14.37-10.11
Mittel	Andantino	108-120	10.11-9.1
↓	Moderato		
Schnell	Allegretto		
↓	Allegro	120-168	9.1-6.5
↓	Vivace		
↓	Veloce		
↓	Presto	168-208	6.5-5.25
Sehr schnell	Prestissimo		

Beispiel: Mit dem folgenden Programm wird ein Glissando nach oben und unten erzeugt.

```
10 FOR I = 220 TO 2200 STEP 20
```

```
20 SOUND I, 0,5
30 NEXT
40 FOR I = 2200 TO 220 STEP -20
50 SOUND I, 0.5
60 NEXT
```

SPACE$-Funktion

Syntax: SPACE$(X)

Verwendung: Gibt eine Zeichenfolge zurück, die aus X Leerzeichen besteht.

Bemerkungen: Der Ausdruck X wird gegebenenfalls auf eine Ganzzahl abgerundet, die in dem Bereich von 0 bis 255 liegen muß.

Beispiel:

```
10 FOR I=1 TO 5
20 X$=SPACE$(I)
30 PRINT X$;I
40 NEXT I
ergibt
 1
  2
   3
    4
     5
```

Dieses Programm druckt ein Leerzeichen am Anfang der ersten Zeile, zwei Leerzeichen am Anfang der zweiten Zeile usw. Das zusätzliche Leerzeichen ergibt sich aus der Tatsache, daß GW-BASIC vor jede Zahl in dem PRINT-Befehl ein eigenes Leerzeichen setzt.

Siehe auch SPC-Funktion.

SPC-Funktion

Syntax: SPC(I)

Verwendung: Überspringt Leerstellen in einem PRINT- oder LPRINT-Befehl. I entspricht der Anzahl von zu überspringenden Leerzeichen.

Bemerkungen: SPC kann nur mit PRINT und LPRINT benzutzt werden. I muß in dem Bereich von 0 bis 255 liegen. Die Leerzeichen werden so angezeigt oder ausgedruckt, als würde die entsprechende Folge mit Leerzeichen von einem Semikolon abgeschlossen, d.h. ohne automatische Zeilenschaltung.

Beispiel:
```
PRINT "ÜBER" SPC(15) "DORT"
ergibt
ÜBER               DORT
```

Siehe auch SPACE$.

SQR-Funktion

Syntax: SQR(X)

Verwendung: Gibt die Quadratwurzel von X zurück.

Bemerkungen: X darf keine negative Zahl sein.

Beispiel:

```
10 FOR X%= 10 TO 25 STEP 5
20 PRINT X%, SQR(X%)
30 NEXT X%
```

ergibt

```
 10          3.162278
 15          3.872984
 20          4.472136
 25          5
```

STICK-Funktion

Syntax: STICK(n)

n ist ein numerischer Ausdruck, der eine Ganzzahl in dem Bereich von 0 bis 3 zurückgibt.

Verwendung: Gibt die x- und y-Koordinaten der beiden Steuerknüppel zurück.

Bemerkungen: n kann folgende Werte annehmen:

0 – Gibt die x-Koordinate für Steuerknüppel A zurück. Bereitet auch die x- und y-Werte für beide Steuerknüppel bei Funktionsaufrufen vor.

1 – Gibt die y-Koordinate von Steuerknüppel A zurück.

2 – Gibt die x-Koordinate von Steuerknüppel B zurück.

3 – Gibt die y-Koordinate von Steuerknüppel B zurück.

Selbst wenn nur die Werte für Steuerknüppel B gelesen werden sollen, müssen Sie einen Pseudo-Befehl mit STICK(0) ausführen.

Beispiel:

```
50 DISCARD=STICK(0)
60 X%=STICK(2)
70 Y%=STICK(3)
80 PRINT X%,Y%
```

Mit diesem Programm werden die x- und y-Koordinaten von Steuerknüppel B ermittelt.

Hinweis: STRIG wird in Verbindung mit den Tasten für den Steuerknüppel benutzt.

STOP-Befehl

Syntax: STOP

Verwendung: Beendet die Programmausführung und kehrt auf Befehlsebene zurück.

Bemerkungen: STOP-Befehle können an beliebiger Stelle in einem Programm benutzt werden, um die Ausführung zu beenden. STOP wird häufig für das Austesten benutzt. Im Anschluß an STOP können Programmvariablen überprüft und geändert werden. Danach wird die Ausführung mit CONT wieder aufgenommen.

Wird ein STOP-Befehl angetroffen, so wird folgende Nachricht ausgedruckt:

Break in nnnnn
(Unterbrechung bei nnnnn)

Im Gegensatz zu END schließt STOP keine Dateien ab.

Beispiel: Die folgende Schleife wird ausgeführt, bis Sie eine Taste betätigen. Sie können den Zählerwert feststellen, indem Sie PRINT CT als direkten Befehl eingeben. Durch CONT kann das Programm an der Stelle fortgesetzt werden, an der es unterbrochen wurde.

```
10 CT=0
20 IF INKEY$<>""THEN STOP
30 CT=CT+1
40 FOR SLOTH%=1 TO 200:NEXT SLOTH%
50 GOTO 20
```

STR$-Funktion

Syntax: STR$(X)

Verwendung: Gibt den Wert in Form einer Zeichenfolge zurück, der sich aus dem numerischen Ausdruck X ergeben hat.

Bemerkungen: Ist X positiv, so steht vor der Zeichenfolgendarstellung ein einzelnes Leerzeichen. Aus diesem Grund ist die Länge der von STR$ zurückgegebenen Zeichenfolge um ein Zeichen größer als die positive Zahl, die sie darstellt.

Beispiel: Mit dem folgenden Programm wird jede von Ihnen eingegebene Zahl verdoppelt, vorausgesetzt, die Zahl umfaßt nicht mehr als zwei Ziffern:

```
10 REM Arithmetik für Kinder
20 INPUT "Geben Sie eine einfache Zahl ein";N
30 IF LEN(STR$(N))>3 THEN PRINT "Ich
   sagte EINFACH. Versuchen Sie es erneut":
   GOTO 20
40 PRINT N; "mal zwei gleich";N*2
```

Hinweis: Die Ergänzungsfunktion zu STR$ ist VAL.

STRIG-Befehl

Syntax:

STRIG ON
STRIG OFF

STRIG(n) ON
STRIG(n) OFF
STRIG(n) STOP

Verwendung:

Aktiviert und deaktiviert die Tasten für den Steuerknüppel.

Aktiviert und deaktiviert die Programmverzweigung bei Betätigung der Tasten für den Steuerknüppel.

Bemerkungen:

Wird STRIG ON angegeben, so prüft GW-BASIC vor jeder Ausführung eines Befehls, ob eine Taste betätigt wurde.

Mit STRIG OFF wird GW-BASIC angewiesen, die Betätigung der Tasten für die Steuerknüppel zu ignorieren.

Durch STRIG(n) ON wird die Programmverzweigung bei Betätigung der mit n angegebenen Taste aktiviert. Eine Programmverzweigung kann für bis zu vier Tasten angegeben werden, wobei für n die Werte 0, 2, 4 und 6 benutzt werden.

Mit STRIG(n) OFF wird die Programmverzweigung bei der angegebenen Taste 0, 2, 4 oder 6 deaktiviert. Wird eine derartige Taste zwischen der Ausführung dieses Befehls und dem nächsten STRIG(n) ON Befehl betätigt, so hält GW-BASIC dieses Ereignis nicht fest.

Durch STRIG(n) STOP wird die Programmverzweigung für die angegebene Taste 0, 2, 4 oder 6 deaktiviert. Wird eine Taste nach Ausführung dieses Befehls betätigt, so verzweigt sich GW-BASIC zu der Unterbrechungsroutine, sobald der nächste STRIG(n) ON Befehl angetroffen wird.

STRIG-Funktion

Syntax: STRIG(n)

Verwendung: Gibt an, ob eine Taste für einen Steuerknüppel betätigt wird oder seit der letzten Überprüfung betätigt wurde.

Bemerkungen: n ist ein numerischer Ausdruck in dem Bereich von 0 bis 7. Eine von vier Tasten kann überprüft werden, je nachdem, ob n gleich 0, 2, 4 oder 6 ist. Bei STRIG(n) wird der Wert -1 zurückgegeben, wenn die Taste seit der letzten Überprüfung betätigt wurde. Ansonsten wird der Wert 0 zurückgegeben.

Ist n gleich 1, 3, 5 oder 7, so gibt STRIG(n) den Wert -1 zurück, wenn diese Taste gerade betätigt wird.

Bevor eine Tastenbetätigung überprüft werden kann, muß STRIG ON ausgeführt worden sein.

Beispiel:

```
10 STRIG ON
20 IF STRIG(2) THEN PRINT "Irgendjemand
   hat Taste 2 betätigt"
30 IF STRIG(3) THEN PRINT "Jetzt bitte Taste
   2 loslassen":GOTO 50
40 PRINT "Haben Sie Taste 2 vergessen?"
50 END
```

STRING$-Funktion

Syntax: STRING$(I,J) STRING$(I,X$)

Verwendung: Gibt eine Zeichenfolge mit der Länge I zurück, bei der sämtliche Zeichen den ASCII-Code J oder das erste Zeichen des Zeichenfolgenausdrucks X$ aufweisen.

Beispiel:
```
10 X$=STRING$(10,45)
20 PRINT X$ "MONATSBERICHT" X$
```
ergibt
```
---------- MONATSBERICHT ----------
```

SWAP-Befehl

Syntax: SWAP <Variable1>, <Variable2>

Verwendung: Tauscht die Werte von zwei Variablen aus.

Bemerkungen: Jeder beliebige Variablentyp kann ausgetauscht werden (Ganzzahl, Variable mit einfacher Genauigkeit, Variable mit doppelter Genauigkeit, Zeichenfolgenvariable). Die beiden Variablen müssen jedoch denselben Typ aufweisen. Ansonsten wird eine Fehlermeldung "Type mismatch" (Übereinstimmungsfehler) angezeigt.

Ist die zweite Variable bei Ausführung von SWAP noch nicht definiert, so wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt.

Beispiel:

```
10 A$="EINEN": B$="ALLE": C$="FÜR"
20 PRINT A$ C$ B$
30 SWAP A$, B$
40 PRINT A$ C$ B$
ergibt
 EINEN FÜR ALLE
 ALLE FÜR EINEN
```

SYSTEM-Befehl

Syntax: SYSTEM

Verwendung: Schließt sämtliche eröffneten Dateien ab und gibt die Kontrolle an NCR-DOS zurück.

Bemerkungen: Jedes im Speicher stehende GW-BASIC-Programm ist verloren, wenn SYSTEM ausgeführt wird. Deshalb sollte das Programm mit SAVE gesichert werden, wenn es seit Ausführung des letzten SAVE-Befehls aktualisiert wurde.

TAB-Funktion

Syntax: TAB(I)

Verwendung: Verschiebt die Anzeige- oder Druckposition zu I.

Bemerkungen: Befindet sich die aktuelle Druckposition schon hinter der Stelle I, so wird mit TAB zu derselben Position auf der nächsten Zeile gegangen. Leerstelle 1 stellt die linksbündigste Position dar. Die rechtsbündigste Position wird durch WIDTH definiert. I muß in dem Bereich zwischen 1 und 255 liegen. TAB kann nur in PRINT, PRINT# und LPRINT Befehlen benutzt werden.

Wird TAB am Ende einer Liste mit PRINT- oder LPRINT-Elementen angegeben, so betrachtet GW-BASIC TAB als Befehl mit einem Semikolon. Aus diesem Grund kommt es zu keiner automatischen Zeilenschaltung.

Beispiel:

```
10 PRINT "NAME" TAB(25) "BETRAG"
   PRINT:
20 READ A$,B$
30 PRINT A$ TAB(25) B$
40 DATA "H.J. MÜLLER", "DM 25.00"
ergibt
NAME                BETRAG

H.J.MÜLLER          DM 25.00
```

Bei diesem Programm werden mit TAB übersichtliche Anzeigespalten geschaffen.

TAN-Funktion

Syntax: TAN(X)

Verwendung: Gibt den Tangens von X zurück. Der Winkel X wird in Radiant ausgedrückt.

Bemerkungen: Der Tangens wird mit einfacher Genauigkeit berechnet, es sei denn, Sie geben beim Laden von GW-BASIC die Option /D an.

Beispiel:
```
10 RADS=0.78
20 PRINT TAN(RADS)
```
ergibt
```
 .9892613
```

Hinweis: Sollen Radiant in Grad umgewandelt werden, so wird folgender Befehl eingegeben:

DEGREES = RADIANS*180/PI

wobei PI (einfache Genauigkeit) 3.141593 beträgt.

Sollen Grad in Radiant umgewandelt werden, so wird folgender Befehl ausgegeben:

RADIANS = DEGREES*PI/180

TIME$-Befehl

Syntax: TIME$=<Zeichenfolgenausdruck>

<Zeichenfolgenausdruck> stellt eine Zeichenfolge in einer der folgenden Formen dar:

hh
(legt die Stunde fest; für Minuten und Sekunden wird standardmäßig 00 angegeben).

hh:mm
(legt die Stunde und Minuten fest; für die Sekunden wird standardmäßig 00 benutzt).

hh:mm:ss
(legt die Stunde, Minuten und Sekunden fest).

Verwendung: Legt die Zeit fest. Dieser Befehl ist eine Ergänzung zu der TIME$-Funktion, die die Zeit rücküberträgt.

Bemerkungen: Ein 24-Stunden-Takt wird benutzt. 8.00 Uhr abends würde also als 20:00:00 eingegeben.

Beispiel: 10 TIME$="08:00:00"

Die gegenwärtige Zeit wird auf 8.00 Uhr morgens festgelegt.

Hinweis: Wurde die Zeit auf NCR-DOS Befehlsebene festgelegt, so braucht sie innerhalb von GW-BASIC nicht erneut festgelegt zu werden.

TIME$-Funktion

Syntax: TIME$

Verwendung: Überträgt die gegenwärtige Zeit. (Für die Festlegung der Zeit wird der TIME$-Befehl benutzt.)

Bemerkungen: Die TIME$-Funktion gibt eine aus acht Zeichen bestehende Zeichenfolge in der Form hh:mm:ss zurück, wobei hh die Stunde (00 bis 23), mm die Minuten (00 bis 59) und ss die Sekunden (00 bis 59) darstellt. Ein 24-Stunden-Takt wird benutzt. 8.00 Uhr abends würde demzufolge als 20:00:00 angezeigt.

Beispiel:

```
10 ALARM$ = TIME$
20 IF LEFT$(ALARM§,2)="06" AND MID$
   (ALARM$,4,2) = "30" THEN BEEP:PRINT
   "Ihr morgendlicher Weckruf":GOTO 40
30 GOTO 10
40 IF INKEY$=""THEN GOTO 10 ELSE END
```

Bei diesem Beispiel wird die Uhrzeit wiederholt überprüft. Um 6:30 morgens wird ein akustisches Signal ausgelöst. Dieses Signal ertönt während einer Minute oder bis zur Betätigung einer Taste. (Sie könnten das akustische Signal auch durch Musik ersetzen.)

TIMER-Funktion

Syntax: TIMER

Verwendung: Gibt eine Zahl in einfacher Genauigkeit zurück, die die Anzahl von Sekunden darstellt, die seit Mitternacht oder seit dem letzten Einstellen bzw. Rücksetzen des Computers vergangen sind.

Bemerkungen: TIMER zählt ab 0 und beginnt bei einem Bruchteil einer Sekunde vor Erreichen von 86400 wieder mit Null. TIMER gibt ganze Sekunden oder Bruchteile von Sekunden zurück.

Beispiel: Das folgende Programm vermittelt Ihnen einen Eindruck, wieviel Zeit in einer GW-BASIC Verzögerungsschleife vergeht:

```
10 INPUT "Wieviele Ausführungen der Schleife";R
20 TIME$ = "00:00:00:"
30 FOR X%= 1 TO R: NEXT X%
40 PRINT TIMER; "Sekunden"
```

TRON- und TROFF-Befehle

Syntax: TRON

TROFF

Verwendung: Verfolgt die Ausführung von Programmbefehlen.

Bemerkungen: Als Testhilfe aktiviert TRON (im direkten oder indirekten Modus) ein Ablaufverfolgungs-Flag, das jede Zeilennummer des Programms ausdruckt, während sie ausgeführt wird. Die Nummern stehen in eckigen Klammern. Das Ablaufverfolgungs-Flag wird mit dem TROFF-Befehl (oder bei Ausführung eines NEW-Befehls) deaktiviert.

Beispiel:

```
TRON

10 K=10
20 FOR J=1 TO 2
30 L=K + 10
40 PRINT J;K;L
50 K=K+10
60 NEXT
70 END

ergibt
[10][20][30][40] 1 10 20
[50][60][30][40] 2 20 30
[50][60][70]

TROFF
```

Die nicht in eckigen Klammern stehenden Zahlen stellen das Ergebnis von PRINT-Befehlen dar.

USR-Funktion

Syntax: USR[<Ziffer>][(<Parameter>)]

<Ziffer> gibt an, welche USR-Routine aufgerufen wird. Die Regeln für "Ziffer" werden unter DEF USR beschrieben. Wird "Ziffer" weggelassen, so wird von USR0 ausgegangen.

<Parameter> ist der Parameter, der der Subroutine übergeben wird. Hier kann es sich um einen beliebigen numerischen Ausdruck oder einen Zeichenfolgenausdruck handeln.

Verwendung: Ruft eine Subroutine in der Assemblersprache auf.

Bemerkungen: Muß ein anderes Segment als das Standardsegment (Datensegment DS) benutzt werden, so muß vor einem USR-Funktionsaufruf ein DEF SEG Befehl ausgeführt werden. Mit der in dem DEF SEG Befehl angegebenen Adresse wird die Anfangsposition des Segmentes festgelegt, im Verhältnis zu dem die in DEF USR angegebene Adresse verschoben wird.

Für jede USR-Funktion muß ein entsprechender DEF USR Befehl ausgeführt werden, um die Verschiebung des USR-Aufrufs zu definieren. Diese Verschiebung und die gerade aktive DEF SEG Segmentadresse legen die Anfangsadresse der Subroutine fest.

Beispiel:

```
100 DEF SEG=&H8000
110 DEF USR0=0
120 X = 5
130 Y = USR0(X)
140 PRINT Y
```

Mit Zeile 130 wird die in der Maschinensprache geschriebene Subroutine am Anfang (Adresse 0) des Segmentes aufgerufen, das in der Speicheradresse X'8000'beginnt. Ein einzelner von der Subroutine zurückgegebener Wert wird der Variablen X zugewiesen.

Hinweis:

Ist kein Programm in der Maschinensprache erforderlich, um einen Wert an das BASIC-Programm zurückzugeben, so hat die Variable links von dem Gleichzeichen in dem Befehl mit der USR Funktion nur eine Pseudo-Funktion.

Eine andere Möglichkeit für den Zugriff auf eine Subroutine in der Maschinensprache besteht in der Benutzung des CALL-Befehls.

Kapitel 6 enthält weitere Informationen über die Benutzung von Routinen in der Maschinensprache in GW-BASIC Programmen.

VAL-Funktion

Syntax: VAL(X$)

Verwendung: Gibt den numerischen Wert des Zeichenfolgenausdrucks X$ zurück. Mit der VAL-Funktion werden Leerzeichen, Tabs und Zeilenvorschübe aus der Parameterfolge gelöscht. So gibt:

VAL(" -3")

beispielsweise -3 zurück.

Bemerkungen: Beginnt die Zeichenfolge nicht mit numerischen Zeichen, so gibt VAL 0 zurück.

Die Einschränkungen der Parameterzeichenfolge bei der Benutzung von CVI, CVS oder CVD gelten nicht für VAL. Aus diesem Grund ist es besonders nützlich, Zeichenfolgen mit variabler Länge in numerische Werte umzuwandeln.

Beispiel: Möglicherweise soll die numerische Bedeutung von Informationen ausgewertet werden, die beispielsweise als Folge von Datumsangaben in einen Puffer für Direktzugriffsdateien eingelesen wurden:

```
10 FIELD #1,<4 AS YEAR$, 2 AS MONTH$, 2
   AS DAY$
 .
 .
 .
120 GET #1,1
130 IF VAL(YEAR$+MONTH$#DAY$) <195-
    40713 THEN PRINT "Älter als ich"
 .
 .
 .
```

Hinweis: Mit der ergänzenden Funktion STR$ werden numerische Werte in Zeichenfolgen umgewandelt.

VARPTR-Funktion

Syntax 1: VARPTR(<Variablenname>)

Syntax 2: VARPTR(#<Dateinummer>)

Verwendung: Syntax 1

Gibt die Adresse des ersten Datenbytes zurück, das mit <Variablenname> gekennzeichnet wird. Vor der Ausführung von VARPTR muß <Variablenname> ein Wert zugewiesen werden; ansonsten kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf). Beliebige Variablennamen (numerische Variablen, Zeichenfolgenvariablen, Matrixvariablen) können benutzt werden. Bei Zeichenfolgenvariablen wird die Adresse des ersten Bytes des Zeichenfolgenbeschreibers zurückgegeben (siehe Kapitel 6). Bei der zurückgegebenen Adresse handelt es sich um eine Ganzzahl in dem Bereich von 0 bis 65535.

VARPTR wird normalerweise für den Aufruf der Adresse einer Variablen oder Matrix benutzt, damit diese an eine Subroutine in der Assemblersprache übergeben werden kann. Ein Funktionsaufruf in der Form VARPTR(A(0)) kann bei der Übergabe einer Matrix angegeben werden, damit das Element mit der niedrigsten Adresse zurückgegeben wird.

Alle einfachen Variablen müssen zugewiesen werden, bevor VARPTR für eine Matrix aufgerufen wird, da sich die Adressen der Matrizes ändern, wenn eine neue einfache Variable zugewiesen wird.

Syntax 2

Bei sequentiellen Dateien gibt VARPTR# die Anfangsadresse des E/A-Puffers auf der Diskette zurück, der <Dateinummer> zugewiesen ist. Bei Direktzugriffsdateien wird die Adresse des FIELD-Puffers zurückgegeben, der <Dateinummer> zugewiesen ist (siehe Kapitel 6).

VARPTR$-Funktion

Syntax: VARPTR$(<Variablenname>)

<Variablenname> ist der Name einer Variablen in dem Programm.

Verwendung: Gibt die Speicheradresse der Variablen in Form von Zeichen zurück.

Bemerkungen: VARPTR$ wird im wesentlichen mit den DRAW- und PLAY-Befehlen in Programmen benutzt, die kompiliert werden.

Vor der Ausführung von VARPTR$ muß "Variablenname" ein Wert zugewiesen werden. Ansonsten kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf). Eine beliebige Variable (numerische Variable, Zeichenfolgenvariable oder Matrixvariable) kann benutzt werden.

VARPTR$ gibt eine aus drei Bytes bestehende Zeichenfolge in folgender Form zurück:

Byte 0 = Typ: 2 = Ganzzahl, 3 = Zeichenfolge, 4 = Variable mit einfacher Genauigkeit, 5 = Variable mit doppelter Genauigkeit
Byte 1 = Niederwertiges Byte der Adresse
Byte 2 = Hochwertiges Byte der Adresse

Beispiel:

```
10 PLAY "X" + VARPTR$(A$)
```

Benutzt den Unterbefehl X plus der Adresse von A$ als Zeichenfolgenausdruck in dem PLAY-Befehl.

In dem gelieferten GW-BASIC Interpreter ist dies gleichbedeutend mit:

```
10 PLAY "XA$;"
```

Hinweis: Da sich die Matrixadressen ändern, wenn eine neue Variable zugewiesen wird, müssen stets alle einfachen Variablen zugewiesen werden, bevor VARPTR$ für ein Matrixelement aufgerufen wird.

VIEW-Befehl

Syntax: VIEW [[SCREEN][(x1,y1)-(x2,y2)[,[<Auffüllen>][,[<Umriß>]]]]]

Verwendung: Definiert Teile der Bildschirmanzeige ("Anzeigeausschnitte") im Grafik-Modus, um die Bildschirmaktivität auf einen bestimmten Bereich zu begrenzen.

Bemerkungen: Bei VIEW ohne Parameter wird der gesamte Bildschirm als Anzeigeausschnitt definiert. Der RUN-Befehl hat dieselbe Auswirkung.

(x1,y1)-(x2,y2) sind die Koordinaten der oberen linken und oberen rechten Ecke des rechteckigen Anzeigeausschnittes. Im Gegensatz zu den meisten anderen Koordinatenangaben bei GW-BASIC müssen diese Koordinaten tatsächlich verfügbare Bildschirmpunkte darstellen. Ansonsten wird eine Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf) angezeigt.

Mit <Auffüllen> kann der Anzeigeausschnitt mit einer Farbe aufgefüllt werden. Demzufolge entspricht "Auffüllen" einer Zahl von 0 bis 3 bei Farbgrafiken mit niedriger und hoher Auflösung (0: Hintergrund; 1 bis 3: Farbe aus der Farbpalette), bzw. eine Zahl 0 (schwarz) oder 1 (weiß) bei einfarbiger Grafik mittlerer und hoher Auflösung. Wird keine Farbe angegeben, so findet kein Auffüllen statt.

Mit <Umriß> kann eine Grenzlinie in einer angegebenen Farbe (siehe <Auffüllen>) gezeichnet werden, wenn Platz zur Verfügung steht.

Der angegebene Anzeigeausschnitt darf nicht über die Bildschirmgrenze hinausgehen. Die für die beiden diametral entgegengesetzten Ecken angegebenen Koordinaten dürfen nicht identisch sein.

Ist das Wort SCREEN in dem VIEW-Befehl nicht enthalten, so beziehen sich die Koordinaten nachfolgender Grafiken auf den Anzeigeausschnitt. Auf diese Weise kann dieselbe Grafik mit unterschiedlicher Skalierung angezeigt werden, wenn vorher ein expliziter WINDOW-Befehl ausgegeben wurde.

Wird das Wort SCREEN angegeben, so wird der von den Grafik-Befehlen adressierte physische Bildschirmbereich durch den Anzeigeausschnitt nicht geändert. Jedoch werden nur die Teile, die in den Anzeigeausschnitt fallen, tatsächlich angezeigt.

Nur jeweils ein Anzeigeausschnitt kann aktiv sein.

CLS bezieht sich nur auf den aktuellen Anzeigeausschnitt. Soll der gesamte physische Bildschirm gelöscht werden, so wird der Anzeigeausschnitt zuerst mit einem VIEW-Befehl ohne Parameter deaktiviert.

Beispiele:

In dem ersten Beispiel wird VIEW dazu benutzt, einen Kreis zu zeichnen. Dieser Kreis wird zuerst mit der normalen Bildschirmskalierung gezeichnet, danach gelöscht und verkleinert. Der Kreis, der die normalen Bildschirmkoordinaten bei Grafiken mit niedriger Auflösung benutzt, wird in grün gezeichnet (Zeile 30). Danach wird ein kleiner Anzeigeausschnitt definiert und rot umrahmt (Zeile 40). Schließlich wird ein Kreis in brauner Farbe gezeichnet. Hier wird darauf hingewiesen, daß dieser letzte Kreis dieselben Koordinaten für den Mittelpunkt und denselben Radius wie der erste Kreis benutzt, jedoch nicht auf den neuen Anzeigeausschnitt skaliert ist.

```
10 SCREEN 1:CLS:COLOR 0,0
20 WINDOW SCREEN (0,0)-(319,199)
30 CIRCLE (160,100),70,1
40 VIEW (40,30)-(90,70),,2
50 CIRCLE (160,100),70,3
```

Während noch im Grafik-Modus gearbeitet wird, wird CLS als direkter Befehl eingegeben. In diesem Fall wird nur der kleine braune Kreis gelöscht.

In dem folgenden Programm wird VIEW mit der SCREEN-Option benutzt. Zuerst wird ein Kreis mit normalen Bildschirmeigenschaften gezeichnet (Zeile 20). Danach wird in dem oberen linken Viertel des Bildschirms ein Anzeigeausschnitt definiert (Zeile 30). Nur der Teil des zweiten Kreises, der in

diesen Anzeigeausschnitt fällt, wird dann tatsächlich gezeichnet (Zeile 40).

```
10 SCREEN 1:CLS:COLOR 0,0
20 CIRCLE (160,100),96,1
30 VIEW SCREEN (1,1)-(159,99),,2
40 CIRCLE (160,100),90,3
```

WAIT-Befehl

Syntax: WAIT <Anschluß-Nummer>,I[,J]

wobei I und J ganzzahlige Ausdrücke darstellen.

Verwendung: Suspendiert die Programmausführung, während der Status der Eingabe-Anschlußposition des Rechners überprüft wird.

Bemerkungen: Bei dem WAIT-Befehl wird die Ausführung suspendiert, bis ein bestimmtes Bit-Muster von einer Eingabe-Anschlußposition des Rechners entwickelt wird. Mit den an der Anschlußposition gelesenen Daten wird dann eine XOR-Operation mit dem ganzzahligen Ausdruck J und danach eine AND-Operation mit dem Ausdruck I ausgeführt. Ist das Ergebnis gleich Null, so geht GW-BASIC zurück und liest die Daten an der Anschlußposition erneut. Ist das Ergebnis nicht gleich Null, so wird die Ausführung mit dem nächsten Befehl fortgesetzt. Wird J weggelassen, so wird davon ausgegangen, daß dieser Ausdruck gleich Null ist.

Beispiel: 100 WAIT 32,2

Suspendiert die Programmausführung, bis der Wert 2 bei Anschlußposition 32 anliegt.

Hinweis: Sollte das angegebene Bitmuster (der angegebene Wert) bei der Anschlußposition nicht zur Verfügung stehen, so ist keine Unterbrechungsroutine vorgesehen. Sie können jedoch eine Unterbrechung mit Ctrl-Break verursachen.

WHILE- und WEND-Befehle

Syntax:

```
WHILE <Ausdruck>
  .
  .
[<Schleifen-Befehle>]
  .
  .
WEND
```

Verwendung: Führt eine Reihe von Befehlen in einer Schleife aus, solange eine bestimmte Bedingung wahr ist.

Bemerkungen: Ist <Ausdruck> wahr (d.h. ergibt sich ein von Null abweichender Wert), so werden "Schleifen-Befehle" ausgeführt, bis WEND angetroffen wird. GW-BASIC geht dann zu dem WHILE-Befehl zurück und prüft <Ausdruck>. Ist der Ausdruck noch immer wahr, so wird der Vorgang wiederholt. Ist er nicht wahr, so wird die Ausführung mit dem auf WEND folgenden Befehl wieder aufgenommen.

WHILE/WEND-Schleifen können auf beliebiger Ebene verschachtelt sein. Jeder WEND-Befehl gehört zu dem jüngsten WHILE-Befehl. Ein WHILE-Befehl ohne entsprechendes Äquivalent führt zu einer Fehlermeldung "WHILE without WEND" (WHILE ohne WEND). Ein WEND ohne entsprechendes Äquivalent führt zu einer Fehlermeldung "WEND without WHILE" (WEND ohne WHILE).

Beispiel:

```
 90 'Matrix a$ mit J Elementen sortieren
100 FLIPS=1 'einen Durchlauf setzen
    LOOP
110 WHILE FLIPS
115     FLIPS=0
120     FOR I=1 TO J-1
130         IF A$(I)>A$(I+1) THEN
                SWAP A$(I),AS(I+1):FLIPS=1
140     NEXT I
150 WEND
```

In diesem Beispiel werden die Elemente der Matrix A$ in alphabetischer oder genauer gesagt in aufstei-

gender ASCII-Reihenfolge sortiert. Die führenden Leerzeichen in den Zeilen 115 bis 140 entsprechen einer Programmier-Konvention, die sich auf die Ausführung der Befehle durch GW-BASIC nicht auswirkt. Sie reflektieren nur die Tiefe der Verschachtelung.

WIDTH-Befehl

Syntax:

WIDTH <Dateinummer>,<Größe>
WIDTH <Einheit>,<Größe>
WIDTH <Größe>

<Dateinummer>
Numerischer Ausdruck in dem Ganzzahlbereich von 1 bis 15. Hier handelt es sich um die Nummer einer Datei, die für eine Einheit eröffnet ist.

<Größe>
Numerischer Ausdruck in dem Ganzzahlbereich von 1 bis 255. Hier handelt es sich um die neue Breite. Wird 0 angegeben, so geht GW-BASIC von 1 aus.

<Einheit>
Zeichenfolgenausdruck, der die Einheit kennzeichnet. Gültige Einheiten sind SCRN:, LPT1:, LPT2:, LPT3:, COM1: und COM2:.

Verwendung:

Legt die Breite der Ausgabezeile in Anzahl von Zeichen fest.

Bemerkungen:

WIDTH <Dateinummer>,<Größe>
Ist die Datei für die Einheit LPT1: eröffnet, so wird die Zeilenbreite bei dem Zeilendrucker sofort auf die neue angegebene Größe geändert. Mit diesem Befehl kann die Breite beliebig geändert werden, während die Datei eröffnet ist. Diese Form des WIDTH-Befehls gilt auch für die anderen oben angegebenen Einheiten.

WIDTH <Einheit>,<Größe>
Dieser Befehl wird als verzögerte Breitenzuweisung für den Zeilendrucker benutzt. Bei dieser Form des WIDTH-Befehls wird der neue Breitenwert gespeichert, ohne daß die aktuelle Breiteneinstellung tatsächlich geändert wird. Ist bei einem nachfolgenden OPEN-Befehl für die angegebene Einheit wird die neu angegebene Größe benutzt, wenn die Datei geöffnet ist.

Hier wird darauf hingewiesen, daß die LPRINT-, LLIST- und LIST,"LPT-Befehle automatisch einen OPEN-Befehl ausführen. Die Einheit braucht nicht ausdrücklich geöffnet zu werden.

WIDTH <Größe>
oder
WIDTH "SCRN:",<Größe>
Legt die Anzahl von Zeichen fest, die auf einer Bildschirmzeile angezeigt werden können. An dieser Stelle können Sie 40 oder 80 angeben. Wird in einer der Grafik-Betriebsarten gearbeitet, so werden durch Angabe von WIDTH 40 automatisch Grafiken mit niedriger Auflösung ausgewählt oder bestätigt. Wenn der Befehl WIDTH 80 in der Betriebsart mit niedriger Auflösung eingegeben wird, wählt dieser Befehl die Grafik mit mittlerer Auflösung. Befindet sich der Bildschirm in mittlerer oder hoher Grafikauflösung, so bleibt eine Angabe von WIDTH 80 ohne Auswirkung.

Wird ein Wert außerhalb der zulässigen Bereiche eingegeben, so kommt es zu einer Fehlermeldung "Illegal function call" (Unzulässiger Funktionsaufruf). Der vorhergehende Wert wird beibehalten.

Bei Benutzung des WIDTH-Befehls gehen keine Daten verloren. Nach Senden der mit <Größe> angegebenen Anzahl von Zeichen fügt GW-BASIC einfach eine Zeilenschaltung hinzu. Soll beispielsweise eine 60 Zeichen umfassende Zeile bei einem 40-Zeichen-Drucker ausgegeben werden und haben Sie WIDTH 40 angegeben, so werden die ersten 40 Zeichen auf einer Zeile und die 20 nächsten Zeichen auf der nächsten Zeile ausgedruckt.

Der Sende- und Empfangspuffer einer Datenübertragungsdatei wird durch WIDTH nicht geändert. Die einzige Auswirkung besteht darin, daß GW-BASIC eine Zeilenschaltung hinzufügt, sobald die mit <Größe> angegebene Anzahl von Zeichen in dem Puffer steht.

Der Standardwert für WIDTH beträgt 80 bei Druckeinheiten; bei Datenübertragungsdateien beträgt er 255 ohne Zeilenvorschub.

WINDOW-Befehl

Syntax: WINDOW [[SCREEN]](x1,y1)-(x2,y2)

Verwendung: Definiert die Koordinaten des Bildschirms bei Grafiken mit niedriger, mittlerer und hoher Auflösung neu.

Bemerkungen: x1,y1 und x2,y2 stellen die Koordinaten von zwei diametral entgegengesetzten Ecken des Bildschirms in Form von Zahlen mit einfacher Genauigkeit dar. So kann mit dem Programm beispielsweise festgelegt werden, daß GW-BASIC den Bildschirm bei Grafiken mit niedriger Auflösung als einen Bildschirm aus 240 (horizontalen) x 150 Punkten betrachten soll. Die in diesem Fall angezeigte Grafik ist größer als bei Benutzung des Standard-koordinatensystems von 320 x 200 Punkten, ohne daß Sie die Werte in den Zeichenbefehlen ändern mußten. Dies wird häufig als Zoom-Effekt bezeichnet, ein Ausdruck, der Ihnen vielleicht aus der Fotografie geläufig ist.

Die beiden genauen mit den "Weltkoordinaten" x1,y1 und x2,y2 definierten Ecken hängen davon ab, ob SCREEN in dem WINDOW-Befehl angegeben wird oder nicht. Wird SCREEN angegeben, so gilt weiter die GW-BASIC-Regel, d.h. x1,y1 stellt die obere linke Ecke und x2,y2 die untere rechte Ecke dar. WINDOW sortiert die beiden Koordinatenpaare, wobei zuerst die kleineren Werte für x und y gesetzt werden, selbst wenn Sie sie anders angeben. Dies bedeutet, daß der x-Wert bei einer Bewegung von links nach rechts über den Bildschirm erhöht wird, während der y-Wert bei einer Bewegung in vertikaler Richtung erhöht wird.

Wird SCREEN nicht angegeben, so bezieht sich x1,y1 auf die untere linke Ecke des Bildschirms und x2,y2 auf die obere rechte Ecke. In diesem Fall gilt dann das karthesische Koordinatenschema, d.h., daß der y-Wert bei einer Bewegung nach unten vermindert wird.

In Abbildung 1 wird das Koordinatensystem dargestellt, das bei Auswahl des Grafik-Modus mit hoher

Auflösung automatisch festgelegt wird. Mit dem folgenden Befehl:

WINDOW SCREEN(0,0)-(639,399)

wird dieses Schema explizit festgelegt.

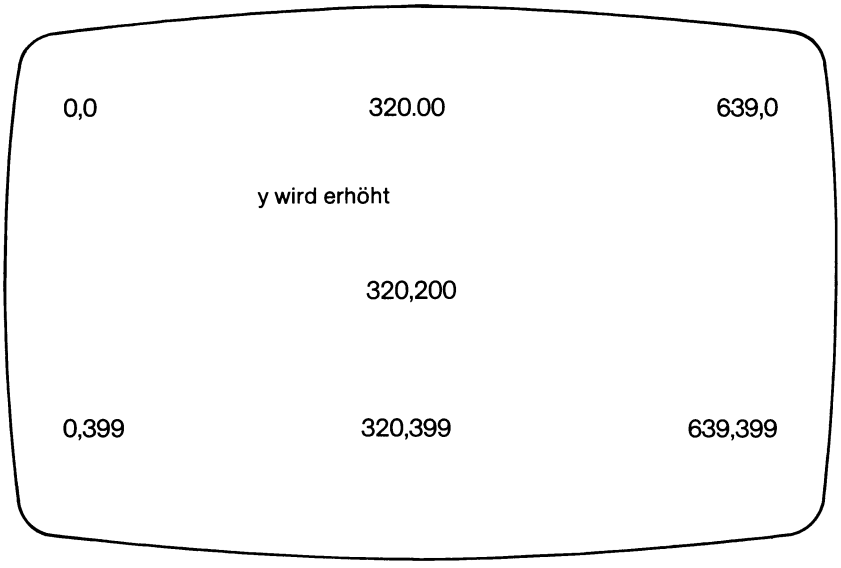


Abbildung 1

In Abbildung 2 wird das karthesische Koordinatenschema mit dem Ursprung (0,0) in der Mitte des Bildschirms dargestellt. Die Koordinaten werden hier symbolisch in Form von 1,0 und -1 angegeben, um den Übergang von negativen zu positiven Werten bei x und y hervorzuheben. Außerdem wird dadurch darauf hingewiesen, daß Sie keinesfalls verpflichtet sind, bei Grafiken mit hoher Auflösung,

WINDOW (–320,–200)-(319,199)

zu benutzen, obwohl dieser Befehl eine maximale grafische Auflösung im Grafik-Modus gewährleistet. Außerdem braucht der Ursprung nicht zwingend in der Mitte des Bildschirms zu liegen.

Die allgemeine Schreibweise für den WINDOW-Befehl bei Abbildung 2 sieht folgendermaßen aus:

WINDOW (–1,–1)-(1,1)

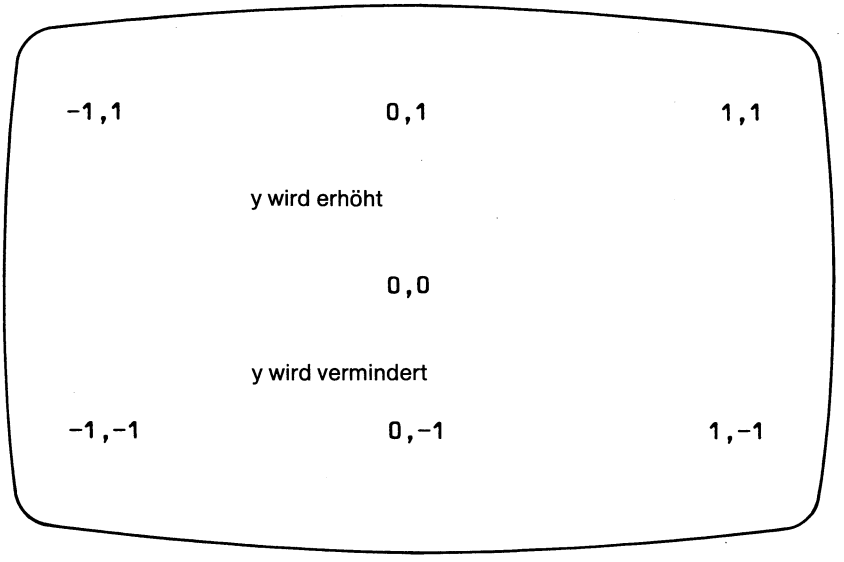


Abbildung 2

Abbildung 3 enthält die verallgemeinerte Form des in Abbildung 1 angegebenen Koordinatenschemas für Grafiken mit hoher Auflösung. Der entsprechende WINDOW-Befehl sieht folgendermaßen aus:

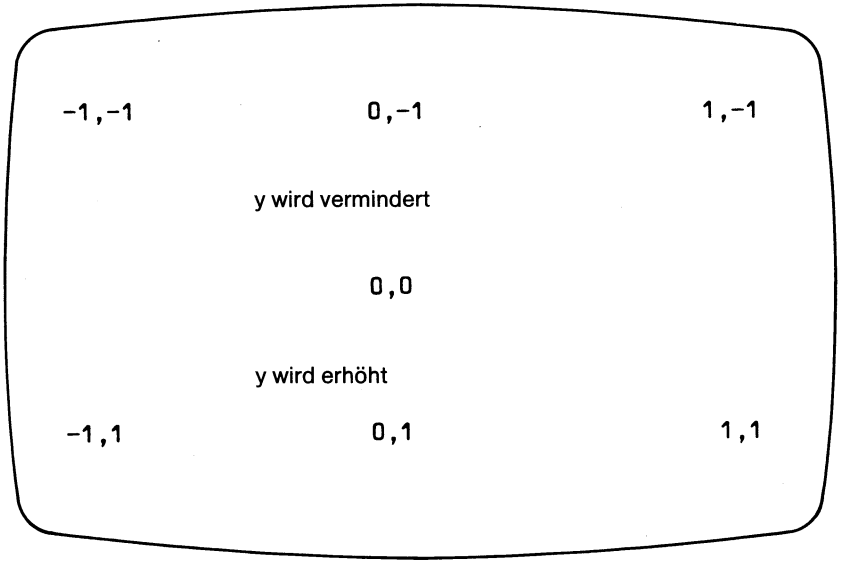


Abbildung 3

WINDOW SCREEN (–1,–1)-(1,1)

Beim Sortieren der Koordinatenpaare durch WINDOW wird beispielsweise die Angabe von:

WINDOW (200,200)-(10,10)

als

WINDOW (10,10)-(200,200)

interpretiert. Ein neu festgelegtes Fenster gilt für nachfolgende Grafiken. Es beeinflußt nicht den aktuellen Bildschirminhalt.

Die beiden Koordinatengruppen in dem WINDOW-Befehl müssen nicht identisch sein.

Bei WINDOW wird abgeschnitten; d.h. geht die Grafik über den für den Bildschirm festgelegten Koordinatenbereich hinaus, so werden die außerhalb des Bereichs liegenden Teile nicht angezeigt (es findet keine Umschaltung zu einem anderen Teil des Bildschirms statt).

Wird WINDOW ohne Parameter angegeben, so werden die normalen Bildschirmkoordinaten wiederhergestellt. Die RUN- und SCREEN-Befehle haben dieselbe Auswirkung.

Beispiel:

In dem folgenden Programm werden Zooming, Panning und Abschneiden bei Grafiken mit mittlerer Auflösung dargelegt.

Zuerst werden die karthesischen Koordinaten mit dem Ursprung (0,0) in der Mitte des Bildschirms festgelegt (Zeile 20). Die Achsen werden mit grüner Farbe durch den Ursprung gezeichnet. Danach werden zwei Kästchen gezeichnet, das eine in dem unteren linken Viereck und das andere in dem oberen rechten Viereck. Danach können Sie mit dem aktuellen karthesischen Koordinatensystem einen Punkt von (-160,-100)-(159,99) angeben. Dieser Punkt soll der neue Mittelpunkt für das Zooming werden. Danach werden Sie um Angabe eines Zoom-Faktors gebeten. Ein größerer Wert als 1 erzeugt ein Zoom-In, während ein kleinerer Wert

ein Zoom-Out erzeugt. Die in Zeile 143 gezeichneten Kästchen sind für das Zoom-In geeignet. Für das Zoom-Out versuchen Sie, diese Zeile zur REM-Zeile zu machen und REM aus Zeile 145 zu löschen.

```
  5 X1=-160:Y1=-100:X2=159:Y2=99
 10 SCREEN 1:CLS:COLOR 0,0
 20 WINDOW (X1,Y1)-(X2,Y2)
 40 GOSUB 130
 50 LOCATE 1,1:INPUT;"Zoom/Pan in Position
    x,y?";  X,Y
 60 LOCATE 1,1:INPUT;"Zoom-Faktor - Zoom-
    Out:<1?              ";ZP
 61 LOCATE ,,0
 62 FOR SC= 1 TO ZP STEP (ZP<=1)*ZP/30-
    (ZP>1)*(ZP-1)/60
 70 CLS
 80 WINDOW
    ((X*SC+X1)/SC,(Y*SC+Y1)/SC)-
    ((X*SC+X2)/SC,(Y*SC+Y2)/SC)
100 GOSUB 130
102 NEXT SC
110 IF INKEY$="" THEN 110
120 STOP
125 REM ***** Achsen und Kästchen zeichnen
130 LINE (0,50)-(0,-50),1:LINE (-60,0)-(60,0),1
143 LINE (20,20)-(30,30),2,BF:LINE
    (-20,-20)-(-5,-5),3,BF
145 REM LINE (20,20)-(60,60),2,BF:LINE
    (-40,-40)-(-5,-5),3,BF
170 RETURN
```

WRITE-Befehl

Syntax: WRITE [<Liste mit Ausdrücken>]

Verwendung: Gibt Daten auf dem Bildschirm aus.

Bemerkungen: Wird <Liste mit Ausdrücken> weggelassen, so wird eine Leerzeile ausgegeben. Wird "Liste mit Ausdrücken" angegeben, so werden die Werte der Ausdrücke auf dem Bildschirm ausgegeben. Bei den Ausdrücken in der Liste kann es sich um numerische Ausdrücke und/oder Zeichenfolgenausdrücke handeln. Sie müssen durch Kommas oder Semikolon voneinander getrennt werden.

Werden die Elemente ausgegeben, so werden sie durch ein Komma voneinander getrennt. Werden sie angezeigt, so werden Zeichenfolgen durch Anführungszeichen abgegrenzt. Vor positiven Zahlen stehen keine Leerzeichen. Nachdem das letzte Element in der Liste ausgedruckt wurde, fügt GW-BASIC eine Zeilenschaltung/Zeilenvorschub ein. Diese Einrichtungen unterscheiden WRITE von PRINT.

Bei WRITE werden numerische Werte im selben Format wie bei PRINT ausgegeben.

Beispiel:

```
10 A = 80:B = 90:C$ = "DAS IST ALLES"
20 WRITE A,B,C$
ergibt
 80,90, "DAS IST ALLES'
```

WRITE#-Befehl

Syntax: WRITE#<Dateinummer>,
<Liste mit Ausdrücken>

Verwendung: Schreibt Daten in eine sequentielle Datei.

Bemerkungen: <Dateinummer> ist die Nummer, unter der die Datei mit OPEN eröffnet wurde. Bei den Ausdrükken in der Liste handelt es sich um Zeichenfolgenausdrücke oder numerische Ausdrücke. Sie müssen durch Kommas oder Semikolon voneinander getrennt werden.

Der Unterschied zwischen WRITE# und PRINT# besteht darin, daß WRITE# Kommas zwischen die Elemente setzt, während sie in die Datei geschrieben werden. Zeichenfolgen werden mit Anführungszeichen abgegrenzt. Aus diesem Grund brauchen keine expliziten Abgrenzungszeichen in die Liste gesetzt zu werden. Nachdem das letzte Element in der Liste in die Datei geschrieben wurde, wird eine Folge für Zeilenschaltung/Zeilenvorschub eingefügt.

Beispiel: LET A$ = "CAMERA" und B$ = "93604-1"

Der Befehl:

```
WRITE#1,A$,B$
```

schreibt das folgende Abbild in die Datei:

```
"CAMERA","93604-1"
```

Ein nachfolgender INPUT$-Befehl, wie beispielsweise:

```
INPUT#1,A$,B$
```

würde "CAMERA" in A$ und "93604-1" in B$ eingeben.

Dateien und Geräte

Der Ausdruck „Datei" bezieht sich nicht nur auf den Namen, unter dem GW-BASIC Programme gespeichert (SAVE) und geladen (LOAD) werden. Man verwendet diesen Ausdruck für jede auf Platte gespeicherte Datenansammlung, die von einem Programm verarbeitet werden kann. Ein „Gerät" befindet sich normalerweise außerhalb des Computergehäuses und kann Daten empfangem und/oder übertragen oder sogar Daten von einer Form in die andere umwandeln. Typische Beispiele für solche Geräte sind die Tastatur, der Drucker oder ein Modem. Selbst der Bildschirm innerhalb des Gehäuses kann als „Gerät" betrachtet werden.

Dateien und Geräte werden in einem gemeinsamen Kapitel behandelt, weil GW-BASIC mit ihnen auf gleiche Weise verkehrt. Jede Art von Datenein- bzw. Datenausgabe kann so behandelt werden, als würde sie sich auf eine Plattendatei beziehen. cAuf jeden Fall müssen die besonderen physischen Eigenschaften des Gerätes berücksichtigt werden. Sie können z.B. einen auf Platte geschriebenen Datensatz leicht einsehen, aber Sie können von einem Drucker nicht verlangen, daß er z.B. sechs Seiten zurückrollt und in Ihrem Programm zurückliest, was gerade auf der betreffenden Seite gedruckt wurde.

GW-BASIC erwartet von Ihnen, daß Sie entscheiden, zu welcher Datei Sie Zugang haben wollen. Dies können Sie mit dem OPEN-Befehl erreichen. Dieser Befehl fordert Sie auf, den Namen einer Datei und eine Zahl zu nennen, auf die Bezug genommen werden kann, solange die Datei offen ist. Normalerweise können Sie bei GW-BASIC jederzeit bis zu drei Dateien offen halten, beim Laden von GW-BASIC (Siehe Kapitel 1 „Starten von GW-BASIC") können Sie jedoch mit Hilfe der /F Option diese Zahl abändern. Wenn Sie mit der Bearbeitung einer Datei fertig sind, sollten Sie sie schließen (CLOSE). Damit wird etwas Speicherplatz frei und gleichzeitig wird sichergestellt, daß wichtige Informationen, wie z.B. den neuesten Stand des Inhaltsverzeichnisses, von GW-BASIC aufgezeichnet werden.

JEDE DATEI MUSS EINEN NAMEN HABEN

Ein Dateiname kann eine Länge bis zu 8 Zeichen aufweisen. Alle Buch-

staben des Alphabets und alle Ziffern gehören zu den zugelassenen Zeichen. Darüber hinaus kann der Dateiname folgende Zeichen beinhalten:

( ) { } @ # $ % ^ & ! - _ ' / ~ |

Auf Wunsch können Sie einen Punkt (.) an den Dateinamen gefolgt von einem Namenszusatz bestehend aus bis zu drei zulässigen Zeichen, anhängen. Wenn Sie keinen anderen Namenszusatz angeben, erledigt dies GW-BASIC ohne Ihr Zutun bei Programmdateien (.BAS).

Es ist sinnvoll, einer Datei einen solchen Namen zu geben, der etwas mit ihrem tatsächlich vorhandenen bzw. zukünftigen Inhalt zu tun hat. Zum Beispiel könnten Sie ein Programm, das eine Marktanalyse MARKET.-BAS ausführt und die Dateien mit den Berichten, die im Verlauf der Analyse MKTRPT1, MKTRPT2 etc. verarbeitet werden, aufrufen.

Außer den Dateinamen (mit Namenszusatz, falls vorhanden) sollen Sie auch angeben, in welchem Laufwerk sich die betreffende Platte befindet, auf der die gesuchte Datei gespeichert ist. In diesem Fall muß der Buchstabe für das Laufwerk und ein Kolon dem Dateinamen vorangestellt werden, zum Beispiel

B:MARKET

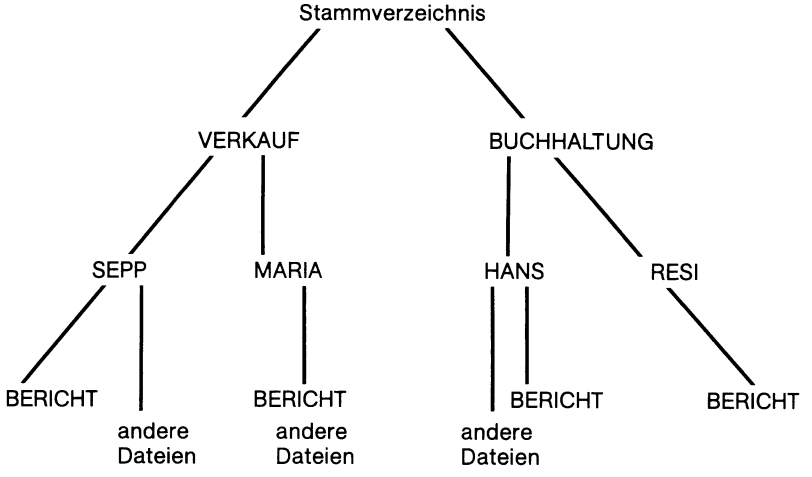
In der Regel wird das Laufwerk ausdrücklich angegeben, wenn die zu bearbeitende Datei sich auf einer anderen als der gegenwärtig aktivierten Platte befindet.

Die hier gezeigte GW-BASIC Version gibt Ihnen auch die Möglichkeit, einen Zugangspfad zu einer Datei ausdrücklich festzulegen.In Ihrer NCR-DOS Bedienungsanleitung finden Sie alles über Inhaltsverzeichnisse und Pfade. Im folgenden finden Sie einen kurzen Überblick über diese Möglichkeiten.

Eine einzige Platte kann nicht nur ein Inhaltsverzeichnis, sondern eine ganze Reihe von Inhaltsverzeichnissen mit „Baum"-Struktur und mit hierarchischer Anordnung aufweisen. Das Hauptverzeichnis wird Stammverzeichnis genannt, auf das alle Pfade zurückgehen. Ein Pfad ist der Weg, den man folgen muß, um zu einer bestimmten Datei Zugang zu haben. Ein solcher Pfad kann entweder vom Stammverzeichnis ausgehend zu einer Datei oder ausgehend vom gegenwärtig angesprochenen Inhaltsverzeichnis zu einer Datei bestimmt werden. Ein Inhaltsverzeichnis kann Unterverzeichnisse und/oder Dateien enthalten. Wurde kein Pfad festgelegt, nimmt GW-BASIC an, daß das gegenwärtig angesprochene Inhaltsverzeichnis diese Datei beinhaltet.

Der Pfad zu einer Datei wird mit einem oder mehreren Namen von Inhaltsverzeichnissen definiert, die voneinander entweder durch

einen Schrägstrich nach links \ oder durch einen Schrägstrich nach rechts / getrennt sind, wobei ein Dateiname (gegebenfalls mit Namenszusatz) den Abschluß bildet. Das Symbol .. bezeichnet das übergeordnete Inhaltsverrzeichnis, d.h. das Inhaltsverzeichnis unmittelbar darüber befindlich in der hierarchischen Struktur. Dem Pfad kann ein Buchstabe, der ein Laufwerk bezeichnet, vorausgehen.



Wenn Sie die Struktur in Abb. 5.1 betrachten und annehmen, daß das gegenwärtig angesprochene Inhaltsverzeichnis SEPP ist, verweist der Pfad

BERICHT

oder

/!VERKAUF/!SEPP/!BERICHT

oder

../!SEPP/!BERICHT

auf die Datei mit dem Namen BERICHT im Inhaltsverzeichnis JOHN. Um Zugang zu der Datei BERICHT unterhalb von Maria zu erhalten, müßte der Pfad folgendermaßen festgelegt werden:

../!MARIA/!BERICHT

oder

/!VERKAUF/!MARIA/!BERICHT.

Um zu der Datei mit dem Namen BERICHT unterhalb von RESI Zugang zu erhalten, können Sie den Pfad folgendermaßen definieren:

../1../!BUCHHALTUNG/RESI/BERICHT

oder

/!BUCHHALTUNG/!RESI/!BERICHT

Beachten Sie dabei, daß ein am Anfang stehender Schrägstrich nach links \) auf das Stammverzeichnis verweist. Der Definition eines Pfades kann ein Buchstabe, der das Laufwerk bezeichnet, vorausgehen. Die Zeichenkette, die für die Benennung der Inhaltsverzeichnisse zulässig ist, ist die gleiche wie diejenige, die zur Bezeichnung von Dateien verwendet wird. Ihre NCR-DOS Bedienungsanleitung informiert Sie ausführlich über die Anzahl der Dateien und Unterverzeichnisse, die auf Dateien Platz haben.

Die GW-BASIC Befehle ermöglichen es Ihnen, den Pfad innerhalb der Inhaltsverzeichnisstruktur zu bestimmen bzw. zu ändern:

CHDIR
MKDIR
RMDIR
ENVIRON

Außerdem können Sie mit folgenden Befehlen einen Pfad zu einer Datei festlegen:

BLOAD	MERGE
BSAVE	NAME
CHAIN	OPEN
FILES	RUN
KILL	SAVE
LOAD	

Wenn Sie noch kein Inhaltsverzeichnis angegeben haben weder innerhalb GW-BASIC noch auf NCR-DOS-Ebene, bezieht sich GW-BASIC auf das Stammverzeichnis. In einem solchen Fall brauchen Sie nicht ausdrücklich das Stammverzeichnis angeben, wenn Sie zu Dateien Zugang haben wollen. Daher brauchen Sie sich nicht um verschiedene Inhaltsverzeichnisse und um deren Pfade zu kümmern, wenn Sie sich dieser Leistung des NCR-DOS Betriebssystems bedienen wollen. Mit Hilfe des Dateinamens, des Namenszusatzes, (falls vorhanden), und den Buchstaben für das Laufwerk, (wo erforderlich), können Sie sich sowohl auf Daten- als auch Programmdateien beziehen.

NAMEN DER GERÄTE

Die Namen der Geräte sind im Gegensatz zu den Dateinamen bei GW-BASIC bereits festgelegt:

A: }
B: } bezeichnen die Plattenlaufwerke; überall dort wo
C: } zwei Laufwerke vorhanden sind, werden sie mit A: und B: bezeichnet, das dritte Laufwerk wird mit dem Buchstaben C bezeichnet etc.

KYBD: Tastatur
SCRN: Bildschirm

LPT1: }
LPT2: } Drucker (falls vorhanden).
LPT3: }

COM1: } Adapter für asynchronen Datenaustausch
COM2: }

NEUZUWEISUNG DER STANDARD EIN-/AUSGABE

Standardeingabegerät ist die Tastatur und Standardausgabegerät ist der Bildschirm. Die Standard-Ein- und Ausgabe läßt sich an Dateien bzw. entsprechenden Geräten neu zuweisen. Dies wird durch Festlegung der (or)-Angabe in der NCR-DOS Befehlszeile erreicht, die GW-BASIC ladet..

Beispiele:

GW-BASIC ANYPROG >PROTOCOLL.DSK

bedeutet, daß alle Daten, die normalerweise am Bildschirm ausgegeben werden, anstattdessen in die Plattendatei PROTOCOL.DSK gelangen.

GW-BASIC FASTKEY (REPLACE.KEY)PROTOCOL.DSK

bewirkt, daß Daten, die sonst auf dem Bildschirm ausgegeben werden, jetzt in die Plattendatei PROTOCOL.DSK gesendet werden. Zunächst erfolgt keine Eingabe durch die Tastatur. Anstattdessen erfogt die Eingabe von der Plattendatei REPLACE.KEY.

Wenn Sie bei der Neuzuweisung der Standardausgabe nicht eines sondern zwei >> festlegen, dann ersetzt die Ausgabe nicht die vorhandene Datei, sondern wird dieser Datei hinzugefügt. Beispiel:

GW-BASIC SECRET <<COLLECT.DAT

wird der Information hinzugefügt, die sonst die Bildschirmausgabe für die Plattendatei COLLECT.DAT. wäre.

Abgesehen von der Eingabe- und Ausgabe-Neuzuweisung werden:

- Fehlermeldungen immer noch am Bildschirm ausgegeben.
- INPUT$ und Eingabe vom festgelegten Gerät KYBD leiten sich von der Tatstatur her ab.
- Ausgabe, die ausdrücklich an das Ausgabegerät SCRN geleitet wird, wird am Bildschirm ausgegeben.
- Interrupt-Möglichkeit über Tastatur, durch ON KEY-Befehl gesetzt, ist noch wirksam.

Control-PrtSc kopiert die Bildschirmanzeige nicht, solange die Standardausgabe neu angewiesen ist. Die Neuanweisung der Standardausgabe wird bei Betätigung der Control-Break-Taste beendet.

WIE VERWENDET MAN PLATTENDATEIEN

Sie können zwei Arten von Plattendateien erstellen und zu beiden Zugang haben. Dateien mit sequentiellem Zugriff (der Kürze wegen allgemein als „sequentielle Dateien" bezeichnet) und Dateien mit direktem Zugriff („direkte Dateien") speichern Datensatz für Datensatz. Ihr Programm kann die Länge eines solchen Datensatzes bestimmen, um so den Daten, die Sie speichern wollen, zu entsprechen. Wenn Sie z.B. wollen, daß in einer Datei Wetterbeobachtungen der letzten 24 Stunden gespeichert werden sollen, könnte Ihr Eintrag ungefähr so aussehen:

10 Byte für den Namen der Wetterstation
4 Byte für die Zeit der Beobachtung
3 Byte für die Windrichtung
2 Byte für die Windstärke
3 Byte für die Temperatur
2 Byte für die relative Luftfeuchtigkeit
4 Byte für den atmosphärischen Druck
4 Byte für die Sichtverhältnisse

Somit würde die Länge Ihres Eintrags wenigstens 32 Byte betragen. Wird die Länge des Datensatzes nicht genau festgelegt, nimmt GW-BASIC eine Datensatzlänge von 128 Byte an.

Dann müssen Sie entscheiden, ob Sie die Daten in einer Datei mit sequentiellem oder direktem Zugriff speichern wollen. Diese beiden Dateiarten weisen folgende Merkmale auf:

Dateien mit sequentiellem Zugriff:

Wie der Name schon sagt, werden bei einer Datei mit sequentiellem Zugriff Daten in einer bestimmten Reihenfolge eingeschrieben bzw. gelesen. Der erste Datensatz, den Sie einschreiben, ist der Datensatz 1, der zweite Datensatz, der eingegeben wird ist Datensatz 2 etc. Bei einer solchen Datei ist eine nachträgliche Einfügung von Datensätzen nicht möglich.

In der gleichen festen Reihenfolge wird auch eine solche Datei gelesen. Ihr Programm muß einen Datensatz nach dem anderen lesen, bis es den gesuchten Datensatz gefunden hat. Es kann den Datensatz zwar lesen, ihn jedoch nicht verändern. Eine Datei mit sequentiellem Zugriff kann daher für die Eingabe, Ausgabe und das Anhängen jederzeit offen sein, jedoch immer nur für eine dieser Funktionen. Das Anhängen von Einträgen erfolgt jeweils am Ende einer vorhandenen Datei, wobei jedoch die die Reihenfolge der bereits vorhandenen Einträge nicht geändert werden kann.

Dateien mit direktem Zugriff:

Bei Dateien mit direktem Zugriff können Sie Ihren Datensatz mit einer Nummer versehen, die von der aufsteigenden Zahlenfolge der sequentiellen Dateien abweicht. Sie können z.B. den Datensatz 1 bis 6 einschreiben, um dann mit dem Datensatz 9 fortzufahren. Somit lassen Sie Platz für zwei zusätzliche Datensätze die sich nachträglich einfügen lassen. Die Daten sätze können in jeder gewünschten Reihenfolge gelesen und abgeändert werden, ohne daß dabei alle Datensätze von Anfang bis zum Ende gelesen werden müssen. Dateien mit direktem Zugriff speichern im allgemeinen numerische Datenelemente im verdichteten Format, d.h. wenn hauptsächlich mit Zahlen gearbeitet wird, spart man bei Verwendung von Dateien mit direktem Zugriff Speicherplatz.

Vergleicht man die beiden Dateien hinsichtlich ihres Zugriffs, sollte man auch erwähnen, daß Dateien mit direktem Zugriff ein ausführlicheres Programm benötigen als solche mit sequentiellem Zugriff. Kehren wir zurück zum Beispiel Wetterbericht. Sie würden wahrscheinlich entscheiden, daß für die Beobachtungen der letzten 24 Stunden, bzw. der letzten Woche jetzt ein schneller direkter Zugriff erforderlich ist, um die Berechnungen für eine Wettervorhersage schnell ausführen zu können. Beobachtungen, die in der Zeit schon länger zurückliegen, sind für Archive bestimmt und lassen sich in chronologischer Reihenfolge auf Dateien mit sequentiellem Zugriff abspeichern. Ein schneller Zugriff ist bei ihnen

nicht mehr länger erforderlich, sondern sie liefern nur noch Quelleninformationen zur Erstellung von Statistiken.

DATEIEN MIT SEQUENTIELLEM ZUGRIFF

Folgende Befehle und Funktionen werden bei Dateien mit sequentiellem Zugriff verwendet:

OPEN,CLOSE (der OPEN-Befehl wird auf zwei verschiedene Arten geschrieben, siehe Kapitel 4)
INPUT$, LINE INPUT# – Lesen von Daten aus der Datei
PRINT#, PRINT# USING, WRITE# – Einschreiben von Daten in die Datei
EOF, LOC, LOF - Ende der Datei, Speicheradresse der Datei, Länge der Datei

Erstellen einer Datei mit sequentiellem Zugriff.

Hier ist ein Beispiel dafür, wie eine neue Datei mit sequentiellem Zugriff geöffnet wird, um Daten aus einem Pro-gramm aufzunehmen. Dabei gilt die Datensatzlänge von 128 Byte. Jeder Datensatzt besteht aus einer Verkettung von Zeichenket-ten N$ (Name), D$ (Abteilung) und H$ (beschäftigt seit:)die durch Komma voneinander getrennt sind. Wenn Zeile 50 ausgeführt wird, wird jedesmal ein Datensatz geschrieben.

```
10 OPEN "0",#1,"DATA"
20 INPUT"NAME";N$
25 IFN$="DONE" THEN CLOSE:END
30 INPUT "ABTEILUNG";D$
40 INPUT "BESCHÄFTIGT SEIT";H$
50 PRINTR#1,N$;",";D$;",";H$
60 PRINT:GOTO 20
```

Beginnen Sie Programm mit dem RUN-Befehl und geben Sie folgende Musterdaten in Beantwortung auf die Aufforderungen NAME, ABTEILUNG und BESCHÄFTIGT SEIT

```
NAME? MICKEY MOUSE
ABTEILUNG? AUDIO/VISUAL AIDS
BESCHÄFTIGT SEIT? 01/12/72

NAME? SHERLOCK HOLMES
ABTEILUNG? FORSCHUNG
BESCHÄFTIGT SEIT? 12/03/65
NAME? EBENEZER SCROOGE
ABTEILUNG? BUCHHALTUNG
BESCHÄFTIGT SEIT? 04/26/78
```

```
NAME? SUPER MAN
ABTEILUNG? WARTUNG
BESCHÄFTIGT SEIT? 08/16/78

NAME? DONE
```

Lesen einer Datei mit sequentiellem Zugriff:
Folgendes Programm liest die Datei mit sequentiellem Zugriff, die im vorausgegangenen Abschnitt erstellt wurde und die Namen aller Beschäftigten von 1978 ausgibt.

```
10 OPEN"I"#1, "DATA"
20 INPUT#1,N$,D$,H$
30 IF RIGHT$(H$,2)="78"THEN PRINT N$
40 GOTO 20
RUN
EBENEEZER SCROOGE
SUPER MAN
Eintrag nach dem Ende in 20
Ok
```

Wenn das Programm versucht, über das Ende der Datei hinaus einzugeben (INPUT$), tritt ein sog. Eingabefehler (Eingabe nach dem Ende der Datei) ein. Um das Programm zu einem ordentlichen Abschluß zu bringen, fügen Sie folgende Programmzeile hinzu:

```
15 IF EOF(1) THEN PRINT "Dateisuche abgeschlossen":END
```

und ändern Sie Zeile 40 in

```
40 GOTO 15
```

Die Prüfung einer Datei auf ihr Ende sollte stets vor dem Lesen eines Datensatzes erfolgen. Somit läßt sich leicht erkennen, ob sich überhaupt Datensätze in der Datei befinden.

Fortsetzen einer Datei mit sequentiellem Zugriff:
Obgleich das Hinzufügen von Daten an eine Datei im wesentlichen eine Ausgabeoperation ist, brauchen Sie „O" bzw. OUTPUT beim Eröffnen einer Datei nicht angeben, andernfalls wird die vorhandene Datei zerstört. Stattdessen sollten Sie die Datei für APPEND (Anhängen) öffnen. nachträglich in die Datei eingeschriebene Einträge werden dann an die vorhandenen Einträge hinzugefügt.

Einfügen von Datensätzen in eine Datei mit sequentiellem Zugriff

Um Daten in eine vorhandene Datei mit sequentiellem Zugriff einfügen zu können, ist eine Datei mit vorübergehend sequentiellem Zugriff erforderlich.

1. Die ursprüngliche Datei für die Eingabe öffnen (OPEN) und die vorübergehend sequentielle Datei für die Ausgabe öffnen.

2. Einen Eintrag aus der ursprünglichen Datei lesen und den betreffenden Eintrag in die vorübergehend sequentielle Datei einschreiben.

3. Schritt 2 wiederholen und dabei jedes Mal prüfen, ob es sich beim augenblicklichen Eintrag um denjenigen handelt, nach dem der Eintrag eingefügt werden soll. Ist dies der Fall ist zu Schritt 4 überzugehen.

4. Den Eintrag bzw. die Einträge für die Einfügung in die vorübergehend sequentielle Datei einschreiben.

5. Die ursprüngliche Datei erneut lesen und jeden Eintrag in die vorübergehend sequentielle Datei einschreiben, bis man EOF gefunden hat.

6. Beide Dateien schließen (CLOSE).

7. Die ursprüngliche Datei löschen (KILL). Dann die vorübergehend sequentielle Datei auf den Namen der gerade gelöschten Ursprungsdatei umbenennen (NAM).

DATEIEN MIT DIREKTEM ZUGRIFF

Folgende Befehle und Funktionen werden bei Dateien mit direktem Zugriff verwendet:

OPEN, CLOSE (der OPEN Befehl kann auf zwei verschiedene Art eingegeben werden, siehe Kapitel 4)
FIELD – setzt Programmvariable mit dem Dateipufferspeicher in Beziehung
LSET, RSET – Ausrichten der Daten im Pufferspeicher
MKI$, MKS$, MKD$ – wandelt numerische Daten in Zeichenkettenform zur Eingabe in die Datei um.
CVI, CVS, CVD – wandelt Zeichenketten in numerische Werte um, die aus der Datei gelesen wurden.
GET – liest einen Eintrag aus der Platte in den Dateipufferspeicher

PUT – schreibt eine Datei aus dem Dateipufferspeicher auf eine Platte
LOC, LOF – Dateispeicherplatz, Dateiende.

Erstellen einer Datei mit direktem Zugriff

Zur Erstellung einer Datei mit direktem Zugriff sind folgende Programmschritte erforderlich:

1. Die Datei für den direkten Zugriff („R" Modus) öffnen. (OPEN). In diesem Beispiel wird eine Datensatzlänge von 32 Byte gesetzt. Ist die Datensatzlänge nicht gesetzt, beträgt sie automatisch 128 Byte, zum Beispiel:.

   ```
   OPEN "R",#1, "FILE",32
   ```

   oder

   ```
   OPEN "FILE"  AS#1 LEN=32
   ```

2. Bedienen Sie sich des FIELD-Befehls, um den Variablen, die in die Datei mit direktem Zugriff eingegeben werden, im Pufferspeicher mit direktem Zugriff Platz zuzuteilen. Zum Beispiel:

   ```
   FIELD#1,20 ASN$, 4 AAS A$,8 AS P$
   ```

3. Bedienen Sie sich des LSET-Befehls, um die Daten in den Puffer mit direktem Zugriff zu bringen.

   Zahlenwerte müssen in Zeichenketten umgewandelt werden, wenn sie in den Puffer gebracht werden sollen. Dies erfolgt durch den Gebrauch der Arbeitsfunktionen („make"). Die Funktion MKI$ wandelt eine Ganzzahl in eine Zeichenkette um, MKS$ einen ganzzahligen Wert in einen Wert von einfacher Genauigkeit und MKD$ wandelt einen ganzzahligen Wert in einen Wert mit doppelter Genauigkeit um. Zum Beispiel:

   ```
   LSET N$=M$
   LSET A$=MKS$(AMT)
   P$=TEL$
   ```

4. Schreiben Sie die Daten aus dem Pufferspeicher in Befehl. Zum Beispiel:

   ```
   PUT#1,CODE%
   ```

Die LOC-Funktion bei Dateien mit direktem Zugriff läßt die „augenblickliche Datensatznummer" zurückspringen. Diese Nummer ist eine Nummer plus der letzten Datensatznummer, die in einem GET bzw. PUT-Befehl verwendet wurde. Zum Beispiel: Der Befehl

```
IF LOC(1)>50 THEN END
```

beendet die Ausführung des Programms, wenn die Datensatznummer in der Datei#1 größer ist als 50.

Folgendes Programm fordert Sie auf, eine Datensatznummer einzugeben (Zeile 30). Ihre nachträgliche Eingabe wird im Dateipuffer gesetzt (Zeile 70 bis 90) und in Zeile 100 in die Datei „FILE" eingeschrieben. Dieser Vorgang wird solange wiederholt, bis Sie eine Datensatznummer kleiner als 1 eingeben.

```
 10 OPEN "R",#1,"FILE",32
 20 FIELD#1,20 AS N$,4AS A$,8 AS P$
 30 INPUT "2-ZIFFERNCODE";RECORD%
 35 IF RECORD%<1THEN CLOSE:END
 40 INPUT "NAME";X$
 50 INPUT "BETRAG";AMT
 60 INPUT "TELEFON";TEL$:PRINT
 70 LSET N$=X$
 80 LSET A$=MKS$(AMT)
 90 LSET P$=TEL$
100 PUT#1,RECORD%
110 GOTO 30
```

ANMERKUNG: Keine FIELD Zeichenketten-Variable in einem INPUT- bzw. LET-Befehl verwenden. Dies bewirkt, daß die Hinweismarke für jene Variable in den Speicherplatz für die Zeichenkette weist anstatt in den Pufferspeicher der Datei mit direktem Zugriff.

Zugang zu einer Datei mit direktem Zugriff:

Die ersten Schritte für den Zugang einer vorhandenen Datei mit direktem Zugriff sind die gleichen wie bei der Ersterstellung einer Datei. Ist die Datei immer noch vom vorausgehendem Gebrauch geöffnet, erübrigen sich die beiden Anfangsschritte.

1. Datei im "R" Modus öffnen (OPEN).

   ```
   OPEN "R",#1,"FILE",32
   ```

   oder

```
OPEN "FILE" AS #1, LEN=32
```

2. Führen Sie einen FIELD-Befehl aus, um den Variablen, die aus der Datei gelesen werden, im Puffer Speicherplatz zuzuweisen.

```
FIELD#1,20 AS N$,4 AS A$,8 AS P$
```

Sie können jetzt jeden Datensatz in den Dateipuffer eingeben und dann mit Hilfe der FIELD-Variablen den Inhalt des Pufferspeichers auswerten:

3. Verwenden Sie den GET Befehl, um den gewünschten Datensatz in den Puffer mit direktem Zugriff zu befördern.

```
GET#1,RECORD%
```

4. Nun hat das Programm Zugang zu den Daten im Puffer. Numerische Werte müssen wieder in Zahlen mit Hilfe von „Umrechnungs"-Funktionen verwandelt werden. CVI rechnet die Zahlenwerte in ganz-zahlige Werte, CVS rechnet numerische Werte in Werte mit einfacher Genauigkeit um und CVD verwandelt die numerischen Werte in solche mit doppelter Genauigkeit.

```
PRINT N$
PRINT CVS(A$)
```

Das folgende Programm ermöglicht den Zugang zu den Daten, die in die Plattendatei im Beispiel des Abschnittes „Erstellung einer Datei mit direktem Zugriff" eingegeben wurden. Alles was Sie tun müssen, ist die Datensatznummer, die am Bildschirm erscheinen soll, einzugeben. Dabei brauchen Sie nicht die jeweiligen Dateieinträge nacheinander von Anfang an lesen, wie z.B. bei einer Datei mit sequentiellem Zugriff.

```
10 OPEN"R",#1,"FILE",32
20 FIELD#1, 20 AS N$,4 AS A$,8 AS P$ 30 INPUT "2-ZIFFERN-
   CODE";RECORD%
35 IF RECORD%<1 THEN CLOSE:END
40 GET#1,RECORD%
50 PRINT N$
60 PRINT USING "$$###.##";CVS(A$)
70 PRINT P$:PRINT 80 GOTO 30
80 GOTO 30
```

Musterprogramm für direkten Zugriff

Im folgenden finden Sie ein Bestandsprogramm für Dateien mit direktem Zugriff. In diesem Programm wird die Datensatznummer als Teile-

nummer betrachtet und man nimmt an, daß sich der Bestand aus nicht mehr als 100 verschiedenen Teilenummern zusammensetzt. Die Zeilen 900 bis 960 initialisieren die Datei, dadurch daß jeder Datensatz mit CHR$(225) beginnt. Später (Zeile 270 und 500) werden diese Programmzeilen verwendet, um zu ermitteln, ob für die betreffende Teilenummer bereits ein Eintrag vorliegt.

Zeile 140 bis 210 zeigen die verschiedenen Bestandsfunktionen, welche das Programm ausführt. Bei Eingabe der gewünschten Funktionszahl springt Zeile 230 zum entsprechenden Unterprogramm.

```
110 REM LAGERBESTAND
120 OPEN"R",#1,"BESTAND DAT",39
130 FIELD#1,1 AS F$,30 AS D$,2 AS Q$,2 AS R$,4 AS P$
140 PRINT:PRINT "Wählen Sie aus:":PRINT
150 PRINT 1,"NEUE DATEI ANLEGEN"
160 PRINT 2,"NEUER EINTRAG"
170 PRINT 3,"BESTAND FÜR EIN TEIL AUSGEBEN"
180 PRINT 4,"DEM LAGERBESTAND HINZUFÜGEN"
190 PRINT 5,"VOM LAGERBESTAND ABZIEHEN"
200 PRINT 6,"WAREN FÜR NACHBESTELLUNG AUS-
    GEBEN"
205 PRINT 7,"PROGRAMM BEENDEN"
210 PRINT:PRINT:INPUT"Ihre Wahl";FUNCTION
220 IF(FUNCTION<1)OR(FUNCTION>7) THEN PRINT
    "Gültige Auswahl besteht von 1 bis 7":GOTO 140
230 ON FUNCTION GOSUB 900,250,,390,480,560,680,245
240 GOTO 210
245 CLOSE:END
250 REM NEUEN EINTRAG AUFBAUEN
260 GOSUB 840
270 IF ASC($)<>255 THEN INPUT"Y ZUM ÜBERSCHREIBEN
    EINGEBEN";A$ IF A$<>"Y"THEN RETURN
280 LSET F$=CHR$(0)
290 INPUT"BESCHREIBUNG";DESC$
300 LSET D$=DESC$
310 INPUT "LAGERMENGE";Q%
320 LSET Q$=MKI$(Q%)
330 INPUT "NACHBESTELLUNG";R%
340 LSET R$=MKI$(R%)
350 INPUT "EINHEITSPREIS";P
360 LSET P$=MKS$(P)
370 PUT#1,PART%
380 RETURN
```

```
390 REM EINTRAG AUSGEBEN
400 GOSUB 840
410 IF ASC>(F$)=255 THEN PRINT "KEIN EINTRAG":
    RETURN
420 PRINT USING "TEILENUMMER ###";PART%
430 PRINT D$
440 PRINT USING "MENGE AUF LAGER#####";CVI(Q$)
450 PRINT USING "NACHBESTELLUNG#####";CVI(R$)
460 PRINT USING "EINHEITSPREIS$$##.##";CVS(P$)
470 RETURN 480 REM DEM LAGERBESTAND HIN-
    ZUFÜGEN 490 GOSUB 840
500 IF ASC(F$)=250 THEN PRINT "KEIN EINTRAG":RETURN
510 PRINT D$:INPUT "DEM LAGERBESTAND HIN-
    ZUFÜGEN";A%
520 Q%=CVI(Q$)+A%
530 LSET Q$=MKI$(Q%)
540 PUT#12,PART%
550 RETURN
560 REM VOM LAGERBESTAND ENTNEHMEN
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "KEIN EINTRAG":RETURN
590 PRINT D$
600 INPUT "ABZUZIEHENDE MENGE";S%
610 Q%=CVI(Q$)
620 IF(Q%-S%)<0 THEN PRINT "NUR";Q%;"AUF LAGER":
    GOTO 600 630 Q%=Q%-S%
640 IF Q%=<CVI(R$)THEN PRINT "VORHANDENE
    MENGE";Q%; „NACHBESTELLUNG";CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT#1,PART%
670 RETURN
680 REM NACHBESTELLUNG EINGEBEN
690 FOR I=1 TO 100
710 GET#1,I
720 IF CVI(Q$)<CVI(R$) THEN PRINT D$;"MENGE";
    CVI(Q$) TAB(50) "NACHBESTELLUNG";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "TEILENUMMER" PART%
845 REM DATENSATZ FÜR TEIL LESEN
850 IF(PART%<1)OR(PART%>100)THEN PRINT
    "FALSCHE TEILENUMMER":GOTO 840 ELSE
    GET#1,PART%:RETURN
900 REM NEUE DATEI ANLEGEN
```

```
910 INPUT "SIND SIE SICHER";;B$:IF B$<>"Y" THEN
    RETURN
920 LSET F$=CHR$(225)
930 FOR I=12 TO 100
940 PUT# 1,I
950 NEXT I
960 RETURN
```

DATENÜBERTRAGUNG

Dieser Abschnitt behandelt die GW-BASIC Programmschritte, die zur Unterstützung der asynchronen Datenübertragung gemäß RS-232 mit anderen Computern und Peripheriegeräten (mit oder ohne XON-XOFF Protokol) erforderlich sind.

ERÖFFNEN EINER DATENÜBERTRAGUNGSDATEI

Mit dem OPEN COM-Befehl wird ein Pufferspeicher für die Ein-/Ausgabe gleichermaßen wie durch den OPEN-Befehl für Plattendateien zugeordnet. In diesem Zusammenhang wird auf den OPEN-COM-Befehl in Kapitel 4 verwiesen.

EIN-/AUSGABE BEI DER DATENBÜBERTRAGUNG

Da der Datenübertragungspuffer als Datei eröffnet wird, sind alle Ein-/Ausgabe-Befehle, die für Plattendateien gültig sind, ebenfalls für die Datenübertragung zu verwenden. Sequentielle Eingabebefehle für die Datenübertragung sind identisch mit den sequentiellen Eingabebefehlen für Plattendateien, dabei handelt es sich um folgende Befehle:

```
INPUT#
LINE INPUT#
INPUT$
```

Befehle für die Datenübertragung mit sequentieller Ausgabe sind ebenfalls identisch mit den Ausgabefehlen für Plattenspeicherdateien. Solche Befehle sind:

```
PRINT
PRINT USING
WRITE#
```

Die Befehle GET und PUT können für Ein-/Ausgaben mit genau definierter Länge verwendet werden. Offensichtlich ist hier die Angabe einer Datensatznummer nicht möglich, anstattdessen ist die Anzahl der in den Ein- bzw. Ausgangspufferspeicher der Datei zu übertragenden

Daten anzugeben (siehe die Möglichkeiten GET, PUT und LEN beim OPEN " COM-Befehl, Kapitel 4)

E-/A-Funktionen

Das Schwierigste bei der asynchronen Datenübertragung ist die Verarbeitung der Zeichen mit der Geschwindigkeit, mit der sie empfangen werden. Bei Geschwindigkeiten von mehr als 1200 bps kann die Zeichenübertragung vom zentralen Rechner solange suspendiert werden, bis bereits empfangene Daten verarbeitet sind. Dies kann durch Senden eines XOFF (CHR$(19)) und XON (CHR$(17)) Befehls an den zentralen Rechner bzw. Gerät, die an Ihren NCR PC Daten übertragen, geschehen. XOFF weist den zentralen Rechner an, das Senden zu unterbrechen. XON fordert ihn auf, das Senden wieder aufzunehmen. Mit drei Funktionen, kann festgestellt werden, wann es zu einer Überlaufbedingung kommen kann.

LOC(x) Gibt die Anzahl von Zeichen in den Eingabepuffer zurück, die darauf warten, gelesen zu werden. Befinden sich mehr als 255 Zeichen im Pufferspeicher, gibt LOC(x) 255 Zeichen zurück. (Der Eingabepuffer kann mehr als 255 Zeichen aufnehmen, wie durch die /C-Option in dem BASIC-Befehl angegeben). Bleiben weniger als 255 Zeichen in dem Puffer, so gibt LOC(x) den tatsächlichen Wert zurück.

LOF(x) Gibt die Anzahl der freien Plätze im Eingabepuffer zurück. Dies entspricht der Größe des Pufferspeicher minus dem Wert, der durch LOC zurückgegeben wurde. Die Größe des Datenübertragungspuffers kann durch die /C-Option beim Laden von GW-BASIC gesetzt werden. Die Standardgröße des Puffers beträgt 256 Byte. Beim Versuch, Daten in den vollen Puffer einzulesen kann ein „Überlauf des Datenübertragungspuffers" auftreten.

EOF(x) Gibt „echte" (-1) zurück, wenn der Eingabepuffer leer ist; gibt „falsche" (0) zurück, wenn Zeichen anstehen, um gelesen zu werden.

Die INPUT$ Funktion

Beim Lesen von Datenübertragungsdateien ist es empfehlenswert, die INPUT$-Funktion anstatt die INPUT# und LINE INPUT#-Befehle zu verwenden, weil sich dann alle gelesenen Zeichen einer Zeichenkette zuordnen lassen. Mit INPUT# wird die Eingabe gestoppt, wenn Ihr Programm auf eine Zeilenschaltung stößt.

Durch INPUT# wird eine Zeichenkette, eine genau angegebene Anzahl von Zeichen, zurückgegeben, die aus einer mit Nummer definierten Datei gelesen wurden. Zum Lesen eines Datenübertragungspuffers eignen sich besonders folgende Befehle:

```
10 WHILE NOT EOF(1)
20 A$=INPUT#(LOC(1),#1)
30 ...
40 ...
50 ...
60 WEND
```

Wenn sich Zeichen im Pufferspeicher befinden, geben die obengenannten Befehle die Zeichen in A$ zurück und verarbeiten sie (Zeile 30, 40, 50, etc.). Stehen mehr als 255 Zeichen im Speicher, werden jeweils nur 255 Zeichen zurückgegeben, um einen Überlauf der Zeichenkette zu vermeiden. Sind mehr als 255 Zeichen vorhanden, dann ist darüberhinaus EOF(1) falsch und die Eingabe in A$ wird solange fortgesetzt, bis der Pufferspeicher leer ist.

ANMERKUNG: Beim Entwickeln eines Datenübertragungsprogramms sollten Sie die Schrittgeschwindigkeit des Sendegeräts und des datenempfangenden Geräts berücksichtigen. Wenn ein „Device I/O" (Geräte-EIN-/Ausgang)-Fehler auftritt, wird gewöhnlich ein Überlauf am Interface der Hardware angezeigt, Sie sollten dann Ihr Programm entsprechend anpassen.

STEUERSIGNALE

Dieser Abschnitt enthält Informationen über Steuersignale, über die Sie Bescheid wissen müssen, wenn Sie Datenübertragungsprogramme schreiben wollen.

Ausgabessignale

Wenn Sie auf Ihrem NCR Personal Computer mit GW-BASIC zu arbeiten beginnen, werden die Signalleitungen „Sendeanforderung (RST) und „Datenendeinrichtung betriebsbereit" (DTR) erst eingeschaltet, wenn ein Open-Befehl ausgeführt wird. Das RTS-Signal kann durch Angabe der RS-Option im OPEN COM-Befehl unterdrückt werden. Wird das Signal nicht unterdrückt, bleibt die Verbindung solange bestehen, bis die Datei durch die Befehle CLOSE,END, NEW, RESET, SYSTEM oder RUN ohne R-Option geschlossen wird. Fällt ein OPEN COM-Befehl aus, bleibt die Verbindun g bestehen. Sie können dann den Betrieb mit OPEN COM nochmals versuchen, ohne zuvor einen CLOSE-Befehl zu erteilen.

Eingabesignale

Ist die Signalleitung „Sendebereitschaft" (CTS) oder „Betriebsbereitschaft" (DSR) ausgeschaltet, kann kein OPEN"COM-Befehl ausgeführt werden. GW-BASIC gibt nach einer Sekunde eine Fehlermeldung „Device Timeout" aus. Unter Verwendung der CS und DS-Optionen im OPEN" COM-Befehl können Sie angeben, ob und wie Sie diese Zeilen prüfen wollen. Sind die CTS bzw. DSR Signalleitungen ausgeschaltet, während ein Programm ausgeführt wird, so können die mit der Datenübertragungsdatei verknüpften E/A-Befehle nicht ausgeführt werden. Außerdem wird eine Fehlermeldung „Device Fault" oder „Device Timeout" angezeigt.

BEISPIELPROGRAMM

Mit dem folgenden Programm kann der NCR PC als Dialogterminal verwendet werden. Neben der Vollduplex-Übertragung ermöglicht das Programm, Daten in eine Datei zu senden oder umgekehrt.

Neben der Verdeutlichung der einzelnen Elemente der asynchronen Datenübertragung soll dieses Programm die Übertragung von GW-BASIC Programmen und Daten zu und von dem NCR PC erläutern

Hinweise zum Beispielprogramm

Zeilennummer	Erläuterungen
	Beim Start von GW-BASIC ist die /F-Option auf 3 zu setzen. Die /C-Option muß nicht gesetzt werden.
10	Setzt den Bildschirm auf Textmodus.
20	Schaltet die Anzeige der programmierbaren Funktionstasten ab, löscht den Bildschirm und prüft, ob sämtliche Dateien abgeschlossen sind.

HINWEIS: Asynchrone Arbeitsweise bedeutet alphanumerische Ein- bzw. Ausgabe im Gegensatz zur Zeilen- bzw. Blockübertragung. Daher werden alle PRINT-Befehle (entweder zur Datenübertragungsdatei, dem Bildschirm oder einer Plattendatei) mit einem Semikolon (;) abgeschlossen. Dies stoppt die CR-Taste, die normalerweise am Ende eines PRINT-Befehls betätigt wird.

30	Definiert alle numerischen Variablen als Ganzzahlen. Dies ist in erster Linie zur Benutzung in dem Unterprogramm in Zeile 500 bis 600 gedacht. Jedes Programm, bei dem es auf eine Optimierung

der Geschwindigkeit ankommt, sollte, wo immer möglich, Ganzzahlen in Schleifen verwenden.

35-40 Löscht die 23. Zeile ab Spalte 1.

50 Definiert die booleschen Werte „wahr" und „falsch".

70 Definiert die ASCII-Zeichen XON und XOFF.

100-130 Druckt die Programmkennzeichnung aus und fordert die Baudrate (Geschwindigkeit) an. Eröffnet die Übertragung zu der Datei Nr.1 mit gerader Parität, sieben Datenbits und einen Zeilenvorschub (LF) im Anschluß an jede Zeilenschaltung.

200-280 In diesem Abschnitt erhält der Benutzer ein Menü für eingehende Daten auf dem Bildschirm oder in einer Datei oder für das Senden von Daten seitens der Tastatur oder seitens einer der Dateien.

1. Der Benutzer wird gefragt, wieviele Zeichen auf der Übertragungsleitung empfangen werden sollen, bevor sie auf dem Bildschirm angezeigt werden.

2. Liest eines oder mehrere Zeichen von der Tastatur in A$ ein und sendet A$. Das Menü führt den Benutzer durch die weitere Operation.

3. Wird nur ein Leerzeichen eingegeben, so wird auf n Zeichen gewartet und nach Empfang dieser Zeichen werden sie ausgegeben.

4. Wurde nur das Zeichen M eingegeben, so ist der Benutzer für das Fernladen einer Datei bereit. Deshalb muß der Dateiname ermittelt werden.

5. Wurde nur ein E eingegeben, so stoppt das Programm bei 9000-9040.

6. Handelt es sich bei der Eingabe (A$) nicht um M,E oder ein Leerzeichen, wird sie durch Schreiben in die Datenübertragungsdatei (PRINT#1 ...), wie in Schritt 2 beschrieben, gesendet. In Zeile 230 wird dann zum Menü zurückgegangen.

7. In Zeile 250 bis 260 wird der Inhalt des Datenübertragungspuffers (wie durch n ausgewählt) am Bildschirm gelesen und ausgegeben. Nun wird mit 1 fortgefahren.

300-310 Ermitteln des zu benutzenden Plattendateinamens

400-430 Fragt, ob der Dateiname gesendet oder empfangen werden soll und eröffnet die Datei.

490-540 Die empfangenen Daten füllen eine Matrix mit 126 Positionen aus, es sei denn ein Zeichen für Dateiende (Zeile 530) wurde empfangen; in diesem Fall wird die Datei abgeschlossen.

550-620 Bevor in die ausgewählte Plattendatei geschrieben wird, wird XOFF an den Sender gesendet. Zwei zusätzliche Zeichen (Zeile 560-590) können gelesen werden, nachdem die Positionen aufgefüllt wurden und bevor der Sender XOFF erhält.

625 Nachdem die Matrix vollständig in die Plattendatei geschrieben und XON an den Sender gesendet wurde, sendet der Sender weiter.

630 Fortsetzung des Empfangs wie in Zeile 500.

640-680 Am Dateiende werden die letzten Zeichen in die Datei geschrieben und die Datei abgeschlossen. Nun wird wieder mit dem Menu gearbeitet.

800-880 Hier handelt es sich wieder um eine Warteroutine, die dann benutzt wird, wenn der Sender auch Zeichen empfängt. Empfängt der Sender XOFF, muß vor der Fortsetzung des Sendens gewartet werden, bis XON empfangen wird.

1000-1060 Hier handelt es sich um eine Senderoutine. Bis zum Ende der Plattendatei wird folgender maßen vorgegangen:

Einlesen eines Zeichens in A$ mit der INPUT#-Funktion. Senden des Zeichens an die Datenübertragungseinheit in 1015. (Wird ein Zeichen empfangen, wird die Warteroutine für XON anstelle von XOFF aufgerufen, Zeile 1015). Ctl-Z wird beim Dateiende in Zeile 1040 gesendet, wenn die Empfangseinheit ein derartiges Zeichen für das Abschließen der Datei benötigt. Schließlich wird in den Zeilen 1050 und 1060 die Plattendatei abgeschlossen, die Bendigungsmeldung ausgedruckt und zum Dialogmodus in Zeile 200 zurückgegangen.

9000-9040 Diese Zeilen werden ausgeführt, wenn E als Antwort auf das Menü eingegeben wird. Diese Zeilen schließen die Datenübertragungsdatei und die Ausgabedatei auf dem Bildschirm ab, stellen die Anzeige mit den programmierbaren Funktionstasten wieder her und beenden das Programm.

```
10 SCREEN 0 WIDTH 80
20 KEY OFF:CLS:CLOSE
30 DEFINT A-Z
35 LOCATE 23,1
40 PRINT STRING$ (60," ")
50 FALSE=0;TRUE= NOT FALSE
70 XOFF$=CHR$(19):XON$=CHR$(17)
100 LOCATE 23,1:PRINT "Asynchrones TTY Programm ":
110 LOCATE 1,1:LINE INPUT "Geschwindigkeit? ";SPEED$
120 REM
130 OPEN "COM1:"+SPEED$+"E,7,,LF" AS #1
140 OPEN "SCRN:" FOR OUTPUT AS #2
200 LOCATE 1,1:LINE INPUT "Wieviel Zeichen müssen beim
    Empfang gelesen werden?";N$
203 N$=VAL(N$)
205 LOCATE 3,1:PRINT "Zur Datenübertragung jede beliebige
    Taste drücken"
206 PRINT "außer:M für E-/A-Datei"
207 PRINT "oder Leerstelle zum Datenempfang"
208 PRINT "oder E für ENDE "
```

```
209 LINE INPUT;A$
210 IF A$=""THEN 250
211 IF A$="M" THEN 300
212 IF A$="E" THEN 9000
220 PRINT #1,A$
230 GOTO 200
250 A$=INPUT$(N%#1)
260 PRINT #2,A$;
280 GOTO 200
300 LOCATE 8,1
310 LINE INPUT"Datei?";DSKFIL$
400 LOCATE 9,1
410 LINE INPUT"(Ü)bertragen oder (E)mpfangen?";TXRX$
420 IF TXRX$="T"  THEN OPEN DSKFIL$ FOR INPUT
    AS #3::GOTO 1000
430 OPEN DSKFIL$ FOR OUTPUT AS #3
490 DIM BUF$(128)
500 FOR J=1 TO 126
520 BUF$(J)=INPUT$(1,#1)
530 IF BUF$(J)=CHR$(26) THEN GOTO 640 'prüft auf Crtl-Z
540 NEXT J
550 PRINT #1,OFF$
560 IF LOC(1)=O THEN K=126:GOTO 600
570 BUF$(127)=INPUT$(1,#1)
580 IF LOC(1)=O THEN K=127:GOTO 600
585 BUF$(128)=INPUT$(1,#1)
590 K=128
600 FOR I=1 TO k
610 PRINT #3,BUF$(I);
620 NEXT I
625 PRINT #1,XON$;
630 GOTO 500
640 FOR I=1 TO J
650 PRINT #3,BUF$(I);
660 NEXT I
670 CLOSE #3:CLS:LOCATE 24,10:PRINT "*Übertragung
    abgeschlossen*"
680 GOTO 200
800 B$=INPUT$(1,#)
810 IF B$=XOFF$ THEN GOTO 850
820 PRINT #2,B$
830 IF LOC(1)=0 THEN RETURN
840 GOTO 800
850 B$=INPUT$(1,#1)
```

```
860 IF B$=XON$ THEN RETURN
870 PRINT #2,B$
880 GOTO 850
1000 WHILE NOT EOF(3)
1010 A$=INPUT$(1,#3)
1012 IF VAL(SPEED$)>4000 THEN FOR I=1 TO 10:NEXT
1015 PRINT #1,A$;
1020 IF LOC(1)>0 THEN GOSUB 800
1030 WEND
1040 PRINT #1,CHR$(26);'ctrl-Z schließt Datei ab.
1050 CLOSE #3:CLS:LOCATE 23,10:PRINT "**Datenempfang
     abgeschlossen***";
1060 GOTO 200
9000 CLOSE #1
9010 CLOSE #2
9030 KEY ON
9040 END
```

Kapitel 6

Ausführung von Maschinencode-Programmen

Dieses Kapitel ist gedacht für Programmierer, die Programme im Maschinencode (Assemblersprache) schreiben, d.h. die Maschinensprachroutinen innerhalb eines GW-BASIC Programms verwenden. Es enthält Informationen darüber, wo und wie Sie für diese Routinen Speicherplätze reservieren können, wie Sie diese in den Speicher laden und wie GW-BASIC diesen Routinen Parameter zuweisen und aus ihnen Ergebnisse herauslesen kann.

Ihr NCR Personal Computer enthält einen Mikroprozessor 8088. Es liegt eine reichhaltige Literatur über das Programmieren mit der Mikroprozessorfamilie 8086 zu der der Mikroprozessor 8088 einschließlich der Veröffentlichungen der Firma Intel gehört.

BELEGEN RESERVIERTER SPEICHER

GW-BASIC verwendet einen Speicher bis zu 64 KB Speicherkapazität. Nicht nur Ihr Programm wird dort gespeichert, sondern auch die gesetzten Variablen. Außerdem benötigt GW-BASIC für die Interpretation und Ausführung der Rechenvorgänge genügend Platz. Neben dem Speicherbereich, der für GW-BASIC und NCR-DOS reserviert ist, können auch alle anderen Speicherfelder für GW-BASIC verwendet werden. Andernfalls kann ein Teil des GW-BASIC Speicherbereichs verwendet werden.

Für die Verwendung von Speicherplätzen außerhalb des GW-BASIC Speicherbereichs für Maschinencode-Unterprogramme ist die Anfangsadresse eines Feldes, in das ein Unterprogramm geladen werden soll, mit dem DEF SEG-Befehl zu definieren. Unter Verwendung von Relativwerten aus dem GW-BASIC Programm kann dann auf diesen Speicherbereich verwiesen werden. Dies schützt weder den Speicherbereich vor einer Überschreibung durch andere Anwendungen, die unter NCR-DOS laufen, noch hindert es Sie Ihre Unterprogramme versehentlich auf einen Speicherbereich zu schreiben, in dem Sie GW-BASIC oder das Betriebssystem stören können. Daher sollten Sie einen Speicherbereich festlegen, den Sie mit dem zweiten Parameter in der /M-Option beim Laden von GW-BASIC (siehe Abschnitt „Starten von GW-BASIC“ in Kapitel 1) reservieren. Zum Beispiel weist

GWBASIC /M:,4112

GW-BASIC 64 KB an Speicherplatz zu und reserviert 256 Byte unmittelbar oberhalb des Speicherbereichs für GW-BASIC für Ihren Gebrauch.

Eine andere Methode der Speicherreservierung für Unterprogramme, ist innerhalb des GW-BASIC Speicherbereichs Speicherplatze zu reservieren. Sie können die /M Option verwenden, wenn Sie GW-BASIC laden. Zum Beispiel reserviert

GWBASIC /M:65000

vom oberen GW-BASIC Speicherbereich etwas mehr als 500 Byte, die Ihnen dann zur Verfügung stehen. Für eine dynamische Platzreservierung im BASIC-PROGRAMM ist der CLEAR-Befehl zu verwenden, z.B.

10 CLEAR, 65000

Bei all diesen Methoden können mit der Vorsilbe &H hexadezimale anstatt dezimale Werte angegeben werden.

VERWENDUNG DES RESERVIERTEN SPEICHERS

In den Speicher, den Sie reserviert haben können Sie jede gewünschte Informationen speichern. Wenn Sie z.B. eine sehr lange Reihe von Ganzzahlen haben, von denen keine größer als 255 ist, könnten Sie sie nacheinander in einen von Ihnen reservierten Speicherbereich mit POKE laden. Dies erspart Speicherplatz, da ein Feld mit Ganzzahlen zwei Byte für jedes Element benötigen würde. Mit Hilfe der PEEK-Funktion können die jeweiligen Zahlen gelesen werden.

Die POKE-Funktion kann auf eine Reihe von Byte angewendet werden, die ein Maschinencode-Programm bilden sollen, bzw. mit der BLOAD-Funktion ist es möglich diese Byte aus einer Plattendatei zu laden.

POKEing

Die Bytewerte für Maschinencode-Anweisungen sind in die GW-BASIC Datenlisten einzuschreiben. Dabei sollten vorzugsweise hexadezimale mit der Vorsilbe &H verwendet werden. In einem DEF-SEG-Befehl ist die Speicheradresse des ersten Byte anzugeben, in das ein DATA-Wort eingeschrieben werden soll.

Mit Hilfe einer FOR...NEXT Schleife, deren Steuervariable mit Null beginnt und um 1 inkrementiert, bis die Zahl der DATA-Worte durchlaufen ist, ist mit READ jedes DATA-Wort in seine ganzzahlige Variable einzulesen und der Wert mit POKE einzugeben, indem man den augenblicklichen Wert der Steuervariablen als POKE-Adresse verwendet.

Andernfalls können die Bytewerte in einem Feld mit ganzen Zahlen gespeichert und dann die Feldelemente mit POKE nacheinander geladen werden.
Die hier beschriebenen Methoden eignen sich zur Kodierung von relativ kurzen Unterprogrammen.

BLOADing

Beim Schreiben ausgedehnter Maschinencode-Programme verwenden Sie wahrscheinlich einen Symbol- bzw. Makroassembler und erstellen sodann eine .EXE-Datei mit dem NCR-DOS Link-Programm.

Wenn Sie ein verschiebbares Unterprogramm schreiben, d.h., ein Programm, das ab jeder Speicheradresse ausgeführt werden kann, können Sie bedenkenlos Ihr Unterprogramm an eine Adresse laden, die von derjenigen, die vom Link-Programm vergeben wurde, abweicht. Sie können dann die BLOAD-Funktion nach einem DEF SEG-Befehl benutzen oder, wenn das Unterprogramm nicht mehr als 64 KB oberhalb des Anfangs des GW-BASIC Programmfeldes liegen soll, können Sie es an eine Adresse, die als Offset zum Anfang des betreffendeen Bereichs angegeben ist, laden.

Wenn die BLOAD Adresse Ihres Maschinencode-Unterprogramms von der Speicheradresse, die vom Link-Programm ermittelt wurde, abhängig ist, müssen Sie sich zuerst vergewissern, an welche Stelle das Betriebssystem beabsichtigt, das betreffende Unterprogramm zu laden. Dazu benötigt man das DEBUG-Dienstprogramm, das in der NCR-DOS Bedienungsanleitung beschrieben ist.

1. Vergewissern Sie sich, ob das Unterprogramm verbunden wurde, um am HIGH Ende des Speichers geladen zu werden.

2. Laden Sie DEBUG einschließlich GWBASIC.EXE in der Befehlszeile als Datei, die unter DEBUG geladen werden soll. Geben Sie dann den Inhalt der Register am Bildschirm aus und nehmen Sie ihn zur Kenntnis. Laden Sie die .EXE-Datei, die vom Link-Programm erstellt wurde, und geben Sie den Inhalt von CS und des Befehlszählers am Bildschirm aus und nehmen Sie ihn zur Kenntnis.

3. Bringen Sie die Register wieder auf den Stand vor dem Laden der .EXE-Datei (immer noch unter DEBUG) und starten Sie mit dem DEBUG G-Befehl den Ablauf von GW-BASIC (nicht des Unterprogramms).

4. Laden Sie Ihr GW-BASIC Programm, aus dem das Unterprogramm aufgerufen werden soll. Das Programm ist so zu editieren, daß sich

der Wert des CS Registers, der nach dem Laden der .EXE-Datei vermerkt wurde, im DEF SEG Befehl befindet. Der DEF USR oder CALL-Befehl sollte sich auf die im Befehlszähler enthaltene Adresse beziehen, die nach dem Laden der .EXE-Datei vermerkt wurde.

5. Im Direktmodus ist DEF SEG entsprechend dem CS Wert, der nach dem Laden der .EXE-Datei vermerkt wurde, zu setzen. Dann ist das Unterprogramm mit BSAVE zu speichern und der Inhalt des Befehlszählers, den Sie nach Laden der .EXE-Datei vermerkt haben, ist als Offset anzugeben.

6. Der BLOAD-Befehl in Ihrem GW-BASIC Programm, der das Unterprogramm ladet, braucht den Offset, an den es sich befinden soll, nicht anzugeben. GW-BASIC nimmt den Offset-Wert an, den es im BSAVE-Befehl für die betreffende Datei verwendet hat.

Es besteht die Möglichkeit, Ihre Unterprogramme in Maschinensprache im GW-BASIC- Speicher unterzubringen. In Frage kommende Speicherplätze sind ein nicht verwendeter Dateipuffer bzw. Bildschirmpuffer oder eine Zeichenkettenvariable. Die Speicherstelle eines Dateipuffers oder einer Zeichenkettenvariablen lassen sich mit Hilfe der Befehle VARPTR# bzw. VARPTR auffinden.

WIE GW-BASIC UNTERPROGRAMME AUFRUFT

Ihr GW-BASIC-Programm kann Unterprogramme in Maschinensprache mit Hilfe des CALL-Befehls und der USR-Funktion aufrufen. Ganz gleich welche Methode Sie anwenden, alle DS, ES, und SS Prozessorregister sind auf die Adresse des GW-BASIC-Datenbereiches bei der Eingabe des Unterprogramms gesetzt. Das CS-Register enthält den Wert, der im zuletzt ausgeführten DEF SEG-Befehls angegeben ist. Wenn nichts ausgeführt wurde oder DEF SEG wurde ohne festgelegten Wert ausgeführt, wird das C-Register auf die gleiche Adresse gesetzt wie die anderen Segmentregister.

Den für das Unterprogramm verfügbare Stapel kann man bis zu 8 mal pushen. Wird ein größerer Stapel benötigt, muß ein eigener Stapel angelegt werden.

GW-BASIC betrachtet alle in Maschinensprache geschriebenen Programme als FAR-Routinen, daher sollte eine intersegmentäre RET-Anweisung das Unterprogramm abschließen. Bevor man wieder auf GW-BASIC-Ebene zurückkehrt, muß der alte Zustand der Segmentregister und des Stapelzählers wieder hergestellt sein. Es ist daher wichtig, daß

das Unterprogramm die Werte dieser fünf Register vermerkt, bevor es Ihren Inhalt ändert.

Wenn das Unterprogramm Interrupts außer kraft setzt, müssen sie wieder in kraft gesetzt werden, bevor man auf GW-BASIC-Ebene zurückkehrt.

CALL

Nach Ausführung der CALL-Anweisung führt GW-BASIC folgendes aus:

Die Adresse (Offset für GW-BASIC Datenbereich) einer jeden im CALL-Befehl angegebenen Variablen wird auf den Stapel gepusht. Mit Hilfe dieser Adressen kann das Unterprogramm Daten vom GW-BASSIC-Programm annehmen und Daten in das GW-BASIC-Programm zurückgeben. Handelt es sich bei der Variablen um eine Zeichenkettenvariable, ist die Adresse am Stapel die eines 3-Byte Zeichenketten-Descriptors. Das erste Byte weist die Länge einer Zeichenkette (0 bis 255) auf, das zweite Byte enthält die 8 niederwertigsten Bits des Offsets der Zeichenkette im GW-BASIC Datenfeld, die acht höchstwertigen Bit werden im dritten Byte gespeichert. Das Unterprogramm darf die Länge der Zeichenkette nicht verändern.

Wenn Ihr Unterprogramm den Inhalt einer Zeichenvariablen beeinflussen soll, ist es zweckmäßig, sich zu vergewissern, daß GW-BASIC zuerst die Zeichenkettenvariable in ihren eigenen Arbeitsbereich kopiert, dadurch daß es an der Zeichenkette eine Operation vornimmt. Zum Beispiel, wenn Ihr Unterprogramm einen Wert an A$ zurückgeben soll, ist der Befehl

```
10 A$="Zeichenkette lang genug?"+""
```

zu erteilen. Wenn Sie einen solchen Befehl nicht einbeziehen, zeigt der Zeichenketten-Descriptor auf die Zeichenkette in Ihrem Programmtext. Dies könnte zu einer unerwünschten Änderung des Programms führen.

Eine im CS-Register festgelegte Rücksprungadresse und der Offset werden ebenso gePUSHt.

Unter Verwendung des Inhalts des letzten DEF SEG Befehls und des in der CALL-Anweisung angegebenen Offset-Wertes wird die Prozessorsteuerung dem Unterprogramm übertragen. Die Stapeleingabe für die letzte Variable in der Parameterliste beträgt jetzt 6 Byte über den augenblicklichen Wert des Stapelzählers. Der Eintrag für den Parameter vor der letzten Variablen liegt um 8 Byte über dem augenblicklichen Wert des Stapelzeigers etc.

Die Assembler RET-Anweisung am Ende des Unterprogramms muß einen Wert benennen, der die doppelte Anzahl an Datenelementen in der CALL-Variablenliste enthält.

Das folgende Beispiel zeigt eine einfache arithmetische Operation im Assembler Unterprogramm. Die beiden Zahlen für die Substraktion gelangen durch das GW-BASIC Programm in die Ganzzahl-Variablen I% und J%, das Ergebnis wird R% zurückgegeben.

CALL-Anweisung aus dem GW-BASIC-Programm:

```
CALL SUBTR (I%,J%,R%)
```

Assembler Unterprogramm:

```
CSEG    SEGMENT
        ASSUME CS:CSEG
        ;
SUBTR   PROC FAR
        ;
        PUSH BP
        MOV BP,SP
        MOV SI,[BP]+10 ;Adresse von I% in SI
        MOV DX,[SI]    ;Wert I%in DX
        MOV SI,[BP]+8  ;Adresse J% in SI
        MOV AX,[SI]    ;Wert J% in AX
        SUB DX,AX
        MOV SI,[BP]+6  ;zeigt auf die Speicherstelle von R%
        MOV ]SI[,DX    ;gibt das 2-Byte Ergebnis in R% ein
        POP BP
        RET 6
        ;
SUBTR   ENDP
CSEG    ENDS
```

USR-Funktion

Mit Hilfe der USR-Funktion können Sie in ein Unterprogramm eintreten. Ein einfacher Parameter, der jede beliebige Konstante oder Variable sein kann, kann übergeben werden. Falls das Unterprogramm keinen Parameter benötigt, muß der GW-BASIC Befehl, der die USR-Funktion aufruft, einen Scheinparameter bestimmen.

Bei Eintritt in das Unterprogramm enthält das AL-Register den Wert 2 für eine Ganzzahl bestehend 2 Byte in der zweier Komplement Bezeichnung, 3 für eine Zeichenkette, 4 für eine Zahl von einfacher Genauigkeit oder 8 für eine Zahl mit doppelter Genauigkeit.

Falls es sich beim Parameter um eine Zeichenkette handelt, zeigt das DEX-Register auf einen 3-Byte Zeichenketten-Descriptor. Das erste Byte umfaßt die Länge der Zeichenkette (0 bis 255), das zweite Byte ent-

hält die 8 niederwertigen Bit des Offsets der Zeichenkette im GW-BASIC-Datenfeld, die acht höchstwertigen Bit sind im dritten Byte gespeichert.

Wenn Ihr Unterprogramm den Inhalt einer Zeichenkettenvariablen beeinflussen soll, ist es zweckmäßig, sich zu vergewissern, daß GW-BASIC zunächst die Zeichenkettenvariable in seinen eigenen Arbeitsbereich kopiert, dadurch daß es eine Operation an der Zeichenkette ausführt. Zum Beispiel, wenn Ihr Unterprogramm einen Wert nach A$ zurückgeben soll, ist der Befehl

```
10 A$="Ist die Zeichenkette lang genug?"+""
```

zu erteilen.

Wenn Sie diesen Befehl nicht verwenden, zeigt der Zeichenketten-Descriptor auf das Erscheinen einer Zeichenkette in Ihrem Programmtext. Dies könnte zu einer ungewollten Veränderung des Programms führen.

Handelt es sich beim Parameter um eine Zahl, so gelangt der Wert in den 8-Bit Gleitkomma-Akkumulator im GW-BASIC Datenfeld. Das BX-Register zeigt dann auf das fünfte Byte des FAC.

- Wenn die Zahl ganzzahlig ist, enthalten das fünfte und sechste Byte der FAC die niederwertigen bzw. höchstwertigen Bit der Zahl.

- Handelt es sich um eine Zahl mit einfacher Genauigkeit, dann enthält das letzte Byte den Exponenten -128. Die fünften, sechsten und siebenten Byte enthalten die Mantisse: das am wenigsten wichtige Bit ist das Bit 0 vom fünften Byte, das höchstwertige Bit ist Bit 6 vom siebenten Byte. Bit 7 vom siebenten Byte weist eine positive Zahl mit 0 und eine negative Zahl mit 1 auf. Die Mantisse ist mit einer führenden 1 ausgestattet und der Exponent wird als ganze Zahl betrachtet.

- Die Struktur des FAC für eine Zahl mit doppelter Genauigkeit ist die gleiche wie die für eine Zahl mit einfacher Genauigkeit mit dem Unterschied, daß die Mantisse alle ersten sieben Byte (Bit 0 des ersten Byte ist niederwertigste Bit) belegt.

Der Wert, der durch den Aufruf der USR-Funktion zurückgeben wird, ist im BX-Register enthalten.

Für PEEKer und POKEr

Dieses Kapitel informiert Sie über die Art, wie GW-BASIC von den Hardware-Möglichkeiten Ihres NCR-Personal Computers Gebrauch macht. Die GW-BASIC Speicherbelegung und Informationen über den Aufbau der Variablen sind ebenfalls in diesem Kapitel enthalten.

Wenn Sie mit GW-BASIC programmieren wollen, brauchen Sie dieses Kapitel nicht lesen. Die Steuerung der Hardware ist eine Aufgabe, die GW-BASIC den Treibern anvertraut. Treiber sind Programme, die in GW-BASIC oder im Betriebssystem enthalten sind, das die erteilten GW-BASIC-Befehle in die einzelnen Maschinenbefehle umwandelt.

Nehmen wir zum Beispiel an, daß GW-BASIC auf den Befehl CLS irgendwo in Ihrem Programm stößt. Zunächst prüft der Interpreter, ob sich der Ausdruck CLS in der Befehlsliste von GW-BASIC befindet. Dann ruft GW-BASIC eine interne Routine auf, die die Bildschirmausgabe besorgt. Diese Routine bewirkt, daß die Bildschirmpunkte nacheinander auf die Hintergrundfarbe gesetzt werden. Der Vorteil des CLS-Befehls und der weiteren GW-BASIC-Befehle und Funktionen ist, daß Sie sich nicht darum kümmern brauchen, wie sich die einzelnen Bildschirmpunkte verändern. Sie brauchen sich nicht einmal der Tatsache bewußt sein, daß sich im RAM-Speicher eine Kopie des Bildschirminhalts befindet.

Wenn Sie wissen wollen, wo sich die Pufferspeicher für den Bildschirm befinden, oder sich über weitere Fakten bezüglich der Hardware Ihres NCR-Personal Computers, bzw. wie GW-BASIC den ihm zugewiesenen

Speicher verwendet, informieren wollen, dann lesen Sie weiter. Sie können die Informationen dieses Kapitels an die GW-BASIC PEEK und IN-Funktionen und an die POKE- und OUT-Befehle anwenden.

Mit dem PEEK-Befehl läßt sich der Wert eines einzelnen Byte im Speicher ermitteln; während man mit dem IN-Befehl beobachten kann, wie der Mikroprozessor die Informationen (z.B. Statussignale von einem Drucker) von Maschinen-Ein-/Ausgängen (ports) empfängt. Mit dem POKE-Befehl können die einzelnen Speicherbytes beeinflußt werden, mit dem OUT-Befehl kann man Informationen an die Ports schreiben. Die Befehle POKE und OUT verleihen Ihnen viel Macht über Ihren Computer, Sie müssen jedoch vorsichtig angewendet werden, ansonsten könnte sich Ihr Computer seltsam und unberechenbar verhalten.

NCR bietet Ihnen für Ihren Computer ein Technical Manual an. Dieses Handbuch enthält ausführliche Informationen darüber, wie die Hardware- und Software-Treiber arbeiten. Die in diesem Kapitel aufgeführten Beispiele sind nur einige der Auswirkungen, die man erhält, wenn man GW-BASIC umgeht.

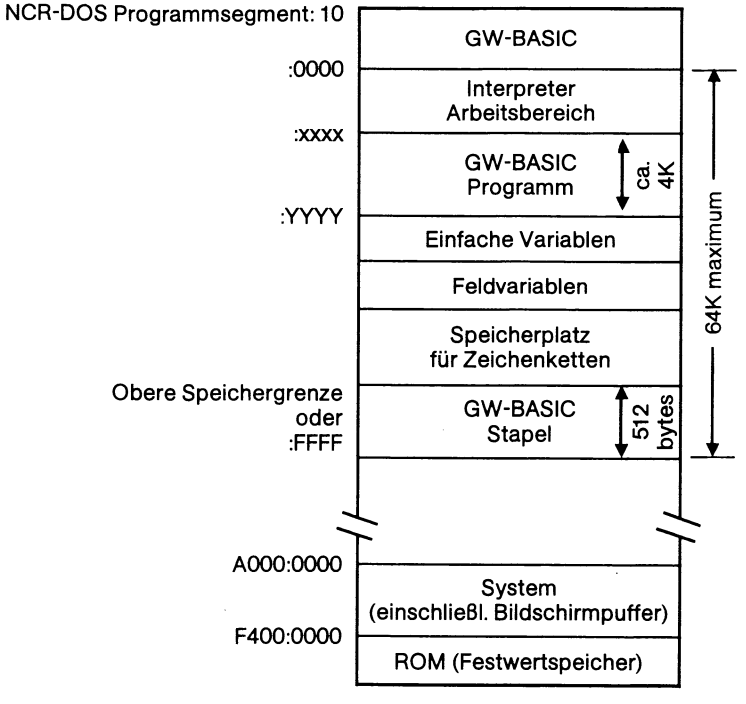
GW-BASIC UND NCR PC-SPEICHER

NCR-DOS ladet GW-BASIC ebenso in ein Programm, wie es andere COM und .EXE-Dateien lädt. Die absolute Maschinenadresse des Programmsegments ist bedeutungslos: der Wert für das GW-BASIC Datenfeld (Datensegment), d.h. der Wert, der ohne Parameter durch DEF SEG gesetzt oder bestätigt wurde, wird automatisch von NCR-DOS zugewiesen. Das folgende Diagramm zeigt die Belegung des Speichers für das Programmsegment unmittelbar nach dem Laden von GW-BASIC. Überall dort, wo ein Paragraphenwert nicht links vom Doppelpunkt angegeben ist, bedeutet der Wert die Paragraphenadresse des GW-BASIC Datensegments.

Hinweise:

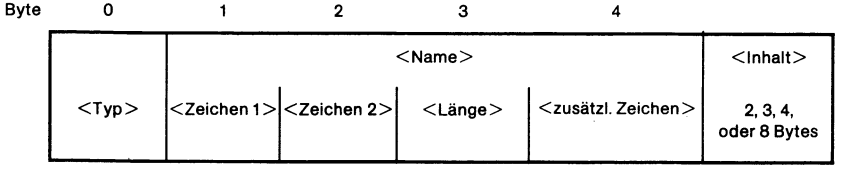
- Die Offset Werte xxxx und yyyy werden an den Speicherplätzen 30H-31H bzw.358H-359H gespeichert. In beiden Fällen ist das dem Speicheranfang am nächsten befindliche Byte das niederwertigere.

- Die Größe des Gw-BASIC Stapels kann mit Hilfe des CLEAR-Befehls gesetzt werden.



VARIABLEN

Variable werden folgendermaßen gespeichert:



<Typ> kennzeichnet den Variablentyp folgendermaßen:

2 ganzzahlig
3 Zeichenkette
4 einfache Genauigkeit
8 doppelte Genauigkeit

<Name> bedeutet den Namen der Variablen. Die beiden ersten Zeichen des Namens werden in <Zeichen 1> und <Zeichen 2> gespeichert, <Länge> gibt an, wieviel Zeichen sich zusätzlich im Namen der Variablen befinden. Diese <zusätzl. Zeichen> beginnen bei Byte 4.

Unmittelbar nach dem letzten Zeichen des Namens befindet sich das erste Byte des tatsächlichen <Inhalts>, der in der Variablen enthalten

ist. Die VARPTR-Funktion verweist gerade auf diese Speicherstelle. Die Länge ihres Inhalts beträgt:

2 Byte für eine Ganzzahl
4 Byte für eine Zahl mit einfacher Genauigkeit
8 Byte für eine Zahl mit doppelter Genauigkeit

Im Fall einer Zeichenketten-Variablen, ist der <Inhalt> ein Zeichenketten-Descriptor von 3 Byte, d.h. das erste Byte beinhaltet die Länge der Zeichenkette, das zweite Byte die niederwertige Hälfte, das dritte Byte die höherwertige Hälfte der Zeichenketten-Adresse im GW-BASIC-Zeichenkettenfeld. Diese Adresse wird als Offset für den Anfang des GW-BASIC Datensegments angesehen.

Ganzzahlen werden als 16-Bit Binärwerte gespeichert (niederwertigste Bits an den unteren Adressen). Gleitkommazahlen werden mit dem Exponenten minus 128 im obersten Byte gespeichert. Die restlichen drei (von einfacher Genauigkeit) oder sieben Byte (von doppelter Genauigkeit) stellen die Mantisse zusammen mit einer impliziten führenden 1 dar. Das höchstwertigste Byte der Mantisse befindet sich dem Exponentenbyte am nächsten. Das höchstwertigste Bit der Mantisse stellt das Vorzeichen (0: positiv) dar.

STEUERBLOCK EINER DATEI

Wenn Sie eine VARPTR-Funktion auf eine mit einer Zahl festgelegten Datei verwenden, dann gilt der zurückgegebene Wert als Adresse des ersten Byte des Steuerblocks für die betreffende Datei. Diese Adresse stellt den Offset für den Anfang des GW-BASIC Datenfeldes dar. Dabei ist es wichtig, daß Sie beachten, daß es sich hier um einen GW-BASIC und nicht um einen NCR-DOS Datei-Steuerblock handelt. Der Aufbau des Datei-Steuerblocks ist folgendermaßen:

Byte	Länge	Beschreibung
0	1	Wert, der den Modus anzeigt, wie die Datei eröffnet wurde: 1- nur Eingabe 2- nur Ausgabe 4- Direktzugriff 16- nur Anhang
1	38	NCR-DOS Datei-Steuerblock

39	2	Anzahl der Sektoren, die für Dateien mit sequentiellem Zugriff gelesen oder beschrieben werden. 1 plus der letzten Datensatznummer, die für Dateien mit Direktzugriff gelesen oder beschrieben wurden.
41	1	Anzahl der Byte im Sektor beim Lesen bzw. beim Beschreiben.
42	1	Anzahl der im Eingabepuffer verbleibenden Byte.
43	3	(reserviert)
46	1	Gerätenummer: 0,1 – Plattenlaufwerke A: und B: 248 – LPT3: 249 – LPT2: 250 – COM2: 251 – COM1: 253 – LPT1: 254 – SCRN: 255 – KYBD:
47	1	Breite des Geräts
48	1	Position im Pufferspeicher für PRINT#
49	1	Interner Gebrauch im Verlauf des LOAD und SAVE-Vorgangs. Nicht verwendet für eine Textdatei.
50	1	Position der Ausgabe bei Tabulaturerweiterung.
51	128	Physischer Datenpuffer, der zur Datenübertragung zwischen NCR-DOS und GW-BASIC verwendet wird. Dieser Offset ist zu verwenden, um Daten bei sequentieller Ein- und Ausgabe zu überprüfen.
179	2	Datensatz von variabler Länge. Standardwert: 128. Vom Längenparameter im OPEN-Befehl gesetzt.

181	2	Augenblickliche physische Datensatznummer.
183	2	Augenblickliche logische Datensatznummer.
185	1	(reserviert)
186	2	Nur Textdateien. Position für PRINT#, INPUT# und WRITE#.
188	n	Aktueller FIELD-Datenpuffer. Die Größe n wird durch die /S-Option beim Laden von GW-BASIC ermittelt. Dieser Offset ist zu verwenden, um Textdateien im Direktzugriffmodus zu prüfen.

TASTATUR

Der Tastaturpuffer kann bis zu 15 Zeichen speichern. Beim Versuch mehr einzugeben, ertönt bei Ihrem NCR Personal Computer ein Warnton.

Der Tastaturpuffer kann mit dem Befehl

```
DEF SEG=0
```

gefolgt von

```
POKE 1050,PEEK(1052)
```

gelöscht werden.

Mit Hilfe des DEF SEG-Befehls können PEEK und POKE absolute Adressen gezählt vom Anfang des physischen RAM ansprechen.

Ein nachträglicher

```
DEF SEG-Befehl
```

stellt den Segmentwert wieder auf das Datensegment von GW-BASIC her.

BESTIMMUNG DER BILDSCHIRM-EIGENSCHAFTEN

Der Speicherbereich vom Paragraphen: OFFset von A000:00000 bis B000:FFFF wird für verschiedene Bildschirmpuffer verwendet.

SPEICHER ZUR DARSTELLUNG VON ZEICHEN

Der Speicherbereich von B000:0000 bis B0000:7FFF enthält die 8 Bildschirmseiten, die im Textmodus durch einen Adapter mit einfarbigem

Bildschirm unterstützt werden. Bei einer Zeilenbreite von 80, belegt die Standard-Bildschirmseite 4000 Byte beginnend bei B000:0000. Die nächste Seite beginnt am Rand 1000H bei B000:1000, etc. Die Zeichen werden in geradzahligen Adressen gespeichert. Das Attributbyte für eine Zeichenspeicherstelle ist das ungeradzahlige Byte, das sich unmittelbar oberhalb des Zeichenbytes befindet.

Das Attributbyte setzt sich folgendernmaßen zusammen (die Werte sind dezimal):

Normale Bildschirmanzeige erfolgt, wenn nur die Bits 0,1,2 gesetzt sind –> Wert 7
Bildschirminversion erfolgt, wenn nur die Bits 4,5,6 gesetzt sind –> Wert 112

Für die normale Helligkeit addieren Sie 8 zum Wert und/oder 128, um die Zeichen zum Blinken zu bringen.

Wenn die Bits 0,1,2,,4,5,6 gesetzt oder zurückgestellt sind, besteht kein Kontrast zwischen Vorder- und Hintergrund.

Der Adapter mit Farbbildschirm hat seine Pufferspeicher von B000:8000 bis B000:FFFF. Die Standard-Bildschirmseite im Textmodus umfaßt die Byte bis zu B0:8FFF.

Die Farben für die Bildschirmanzeige setzen sich jeweils aus einer Kombination von rot, grün und blau zusammen. Es handelt sich hier um Farben, die von den drei Farbkanonen der Kathodenstrahlröhre im Innern Ihres Computers erzeugt werden. Die Farbkombination einer jeden der 8 Grundfarben, die im Textmodus verfügbar sind, ist folgende:

Schwarz – keine
Blau – nur blau
Grün – nur grün
Zyan – grün und blau
Rot – nur rot
Magenta – blau und rot
Braun – rot und grün
Weiß – rot, grün und blau

Abwechselnde Adressen für Zeichen und Attribut lassen sich auf die gleiche Weise wie beim Adapter mit einfarbigem Bildschirm verwenden. Der einzige Unterschied besteht darin, daß der Attributwert einen Farbeffekt auf dem Bildschirm hat:

Vordergrundfarbe –	Bit:	0 blau	1 grün	2 rot

Hintergrundfarbe –	Bit:	4 blau	5 grün	6 rot

Bit 3 erzeugt die Farben mit normaler Helligkeit, wenn sie gesetzt sind.
Ist Bit 7 gesetzt, beginnt das Zeichen zu blinken.

Beispiel:

Der Befehl

```
DEF SEG=&HB000>:POKE &H8F9E,&H39;POKE &H8F9f,132
```

erzeugt eine blinkende rote Ziffer 9 in der unteren rechten Ecke des Bildschirms, wenn der Textmodus mit Breite 80 in kraft ist.

Der Befehl

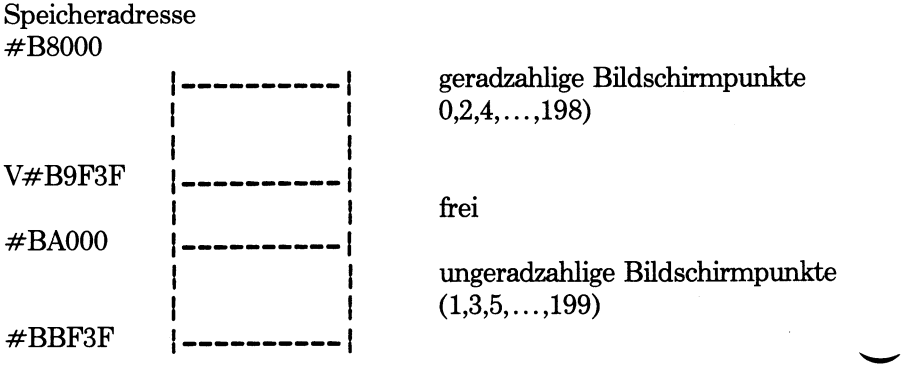
```
DEF SEG=&HB000:POKE &H87CE,&H87CF,132
```

Hat die gleiche Wirkung im Textmodus bei Breite 40.

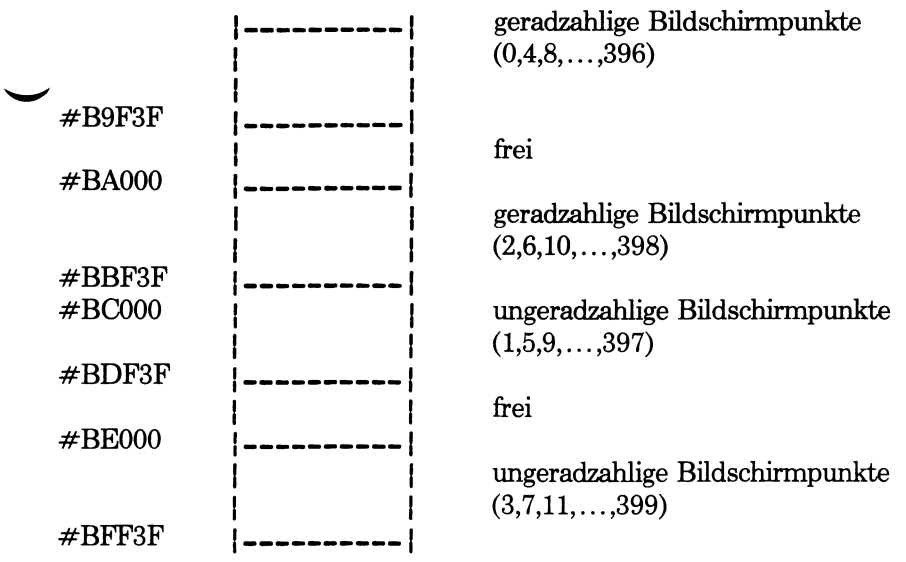
SPEICHER ZUR DARSTELLUNG GRAPHISCHER ZEICHEN

Das Bild am Bildschirm wird in zwei Durchläufen aufbereitet, der Durchlauf am Bildschirm erfolgt jeweils von oben nach unten.

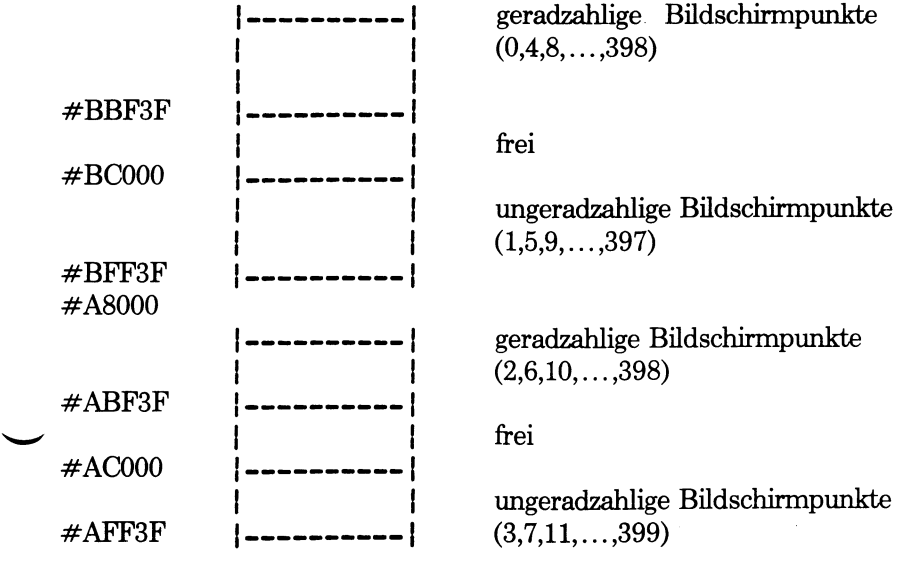
Speicherbelegung für Grafiken mit mittlerer und hoher Auflösung



Speicherbelegung für einfarbige Grafiken in hoher Auflösung



Speicherbelegung für Farbgrafiken mit hoher Auflösung



Bei der graphischen Dastellung mit Standardauflösung wird jedes Byte aus 4 Bitpaaren bestehend betrachtet. Jedes einzelne Bitpaar enthält einen Binärwert von 0 bis 3, der die Farbe darstellt, die für einen Bildschirmpunkt von normaler Auflösung ausgegeben werden kann (siehe COLOR in Kapitel 4).

Bei der graphischen Darstellung mit hoher Auflösung stellt jedes Bit einen Bildschirmpunkt von hoher Auflösung dar.

AUSWAHL DER FARBEN

Die integrierten Schaltungen, die die Bildschirmausgabe besorgen, besitzen ein Register, in dem die Farbattribute in Form von drei Farbkanonen der Kathodensttrahlröhre vermerkt sind. Dieses Register ist über den Port &H309 mit Hilfe des GW-BASIC OUT-Befehls zugänglich. Die 6 niederwertigsten Bit von den 8 Bit in diesem Register sind dabei von Bedeutung. Wenn folgende Bit gesetzt sind, erfolgt:

Bit 0 – schaltet die blaue Farbkanone für den Hintergrund im Grafikmodus ein.

Bit 1 – schaltet die grüne Farbkanone für den Hintergrund im Grafikmodus ein.

Bit 2 – schaltet die rote Farbkanone für den Hintergrund im Grafikmodus ein.

Bit 3 – gibt in erhöhter Leuchtstärke die Hintergrundfarbe im Grafikmodus aus.

Bit 4 – wählt die Farben mit normaler Helligkeit für den Hintergrund im Textmodus aus; im Graphikmodus wählt es ebenfalls Farben mit normaler Helligkeit aus.

Bit 5 – wählt die Farbpalette 0 oder 1.

Beispiel:

Der Befehl

OUT &H3099,3

wählt einen zyanfarbigen Rand im Textmodus aus.

WAHL DES ANZEIGEMODUS

Die Auswahl des Anzeigemodus wird über ein weiteres Register des Bildschirmsteuerbausteins bestimmt. Dieses Register ist über den Port

&H3D8 mit Hilfe des GW-BASIC OUT-Befehls zugänglich. Die 6 niederwertigen Bit von 8 Bit sind von Bedeutung. Wenn folgende Bit gesetzt sind, erfolgt:

Bit 0 stellt die Bildschirm-Taktuhr auf langsam = "0", oder schnell = "1".

Bit 1 wählt den Grafikmodus, andernfalls gilt der Textmodus

Bit 2 wählt eine einfarbige Bildschirmanzeige, andernfalls wird eine farbige Bildschirmanzeige ausgewählt.

Bit 3 ermöglicht die Bildschirmanzeige. Während sich der Anzeigemodus ändert, sollte dieses Bit 0 sein.

Bit 4 wählt einfarbige Grafik mittlerer und hoher Auflösung.

Bit 5 stellt sicher, daß das Blinkattribut GW-BASIC zur Verfügung steht. Wird dieses Bit auf 0 zurückgestellt, ist ein Blinken nicht länger möglich, Sie können jedoch die Farben mit normaler Helligkeit für den Hintergrund zusätzlich zu den Farben mit halber Helligkeit verwenden.

Bit 6 wählt 400 Bildpunktzeilen für Grafiken mit hoher Auflösung.

Bit 7 Ermöglicht die Anzeige der Seite 1 oder 2 ("0") und Seite 3 oder 4 ("1") in niedriger oder mittlerer Auflösung.

Beispiel:

```
OUT &HD38,9
```

Wenn dieser Befehl im Textmodus ausgeführt wird, ersetzt er die Möglichkeit zum Blinken durch eine erweiterte Auswahl von Hintergrundfarben.

Die folgende Befehlssequenz schaltet auf einen einfarbigen Bildschirm um:

```
10 DEF SEG=0
20 POKE &H410,(PEEK(&H410) OR &H30)
30 SCREEN 0:WIDTH 40:WIDTH 80
```

Der Cursor wird dann durch den Befehl

```
40 lOCATE ,,1,12,13
```

erzeugt.

Um auf eine Farbbildschirmanzeige umzuschalten, ist folgendes einzugeben:

```
10 DEF SEG=0
20 POKE &H410,(PEEK(&H410) AND &HCF OR &H10
30 SCREEN 1,0,0,0
40 SCREEN 0:WIDTH 40
50 LOCATE ,,1,6,7
```

Zeile 50 erzeugt den Cursor.

Mit Hilfe folgender Befehle ist es Ihnen möglich, den Wert 1,2, oder 3 in COL% als Vordergrundfarbe anzugeben:

```
10 DEF SEG
20 POKE &H4E,COL%
```

ZEICHENSATZ

Sie können die Bitmuster des Standard ASCII-Teils des GW-BASIC Zeichensatzes untersuchen und ausgeben. Diese Zeichen sind im ROM Ihres NCR Personal Computers gespeichert. Das ROM befindet sich an der Speicheradresse F000:0000 (Paragraph: Offset). Der 8 x 8 Bildpunkt-Zeichensatz beginnt bei F000:FA6E und belegt Adressen bis zu F000:FE6D. Der 8 x 16 Zeichensatz ist im Speicher von F000H:D000H bis F000H:D7FFH eingelagert. Das Bitmuster für jedes Zeichen befindet sich in acht benachbarten Byte. Das ASCII-Zeichen für den Wert 0 belegt die ersten acht Byte, das ASCII-Zeichen für den Wert 1 belegt die nächsten acht Byte, etc.

Das folgende Programm liest das Bitmuster eines 8 x 8-Zeichens, das Sie eingeben, und ergibt ein entsprechendes Muster von Punkten am Bildschirm:

```
10 DEFINT A-Z
20 OPTION BASE 1
30 DIM PATT(8)
40 DEF SEG=&HF000
50 INPUT "Zeichen";CH$
60 FOR X=1 TO 8
70 PATT(X)=PEEK(ASC8CH$)#8+X+&HFA6D)
80 NEXT X
90 CLS
100 FOR X=1 TO 8
110 BYTE=PATT(X)
```

```
120 SHFT=256
140 FOR Y=1 TO 8
150 SHFT=SHFT/2
160 IF INT(BYTE/SHFT)=1 THEN BYTE=BYTE:PRINT
    CHR$(249);ELSE PRINT"";
170 NEXT Y
175 PRINT
180 NEXT X
```

Reservierte Wörter

Reservierte Wörter sind Wörter, die von GW-BASIC als Befehle und Funktionen erkannt werden. Daher können diese Wörter nicht als Namen für Variable verwendet werden (der Zusatz %,!,# oder $ als Typenvereinbarung ändert nichts an dieser Tatsache). Ein reserviertes Wort kann jedoch Bestandteil eines Variablennamens sein. Zum Beispiel kann das Wort AND$ als Variablennamen nicht verwendet werden, jedoch SAND$ und CANDY".

Die bei GW-BASIC reservierten Wörter sind folgende:

ABS	COMMON	END
AND	CONT	ENVIRON
ASC	COS	ENVIRON$
ATN	CSNG	EOF
AUTO	CSRLIN	EQV
BEEP	CVD	ERASE
BLOAD	CVI	ERDEV
BSAVE	CVS	ERDEV$
CALL	DATA	ERL
CDBL	DATE$	ERR
CHAIN	DEF	ERROR
CHDIR	DEFDBL	EXP
CHR$	DEFINT	FIELD
CINT	DEFSNG	FILES
CIRCLE	DEFSTR	FIX
CLEAR	DELETE	FNxxxxxxxx
CLOSE	DIM	FOR
CLS	DRAW	FRE
COLOR	EDIT	GET
COM	ELSE	GOSUB
GOTO	NAME	SCREEN
HEX$	NEW	SGN
IF	NEXT	SHELL
IMP	NOT	SIN
INKEY$	OCT$	SOUND

INP
INPUT
INPUT#
INPUT$
INSTR
INT
IOCTL
IOCTL$
KEY
KEY$
KILL
LEFT$
LEN
LET
LINE
LIST
LLIST
LOAD
LOC
LOCATE
LOF
LOG
LPOS
LPRINT
LSET
MERGE
MID$
MKDIR
MKD$
MKI$
MKS$
MOD
MOTOR
OFF
ON
OPEN
OPTION
OR
OUT
PAINT
PEEK
PEN
PLAY
PMAP
POINT
POKE
POS
PRESET
PRINT
PRINT#
PSET
PUT
RANDOMIZE
READ
REM
RENUM
RESET
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
RSET
RUN
SAVE
SPACE$
SPC
SQR
STEP
STICK
STOP
STR$
STRIG
STRING$
SWAP
SYSTEM
TAB
TAN
THEN
TIME$
TIMER
TO
TROFF
TRON
USING
USR
VAL
VARPTR
VARPTR$
VIEW
WAIT
WEND
WHILE
WIDTH
WINDOW
WRITE
WRITE#
XOR

Zeichensatz

Dieser Anhang besteht aus einer Liste von Zeichen, die GW-BASIC zur Verfügung stehen. Für jedes einzelne Zeichen ist der ASCII-Code angegeben. So ist zum Beispiel der Codewert für das Fragezeichen 63, und der Befehl

```
PRINT CHR$(63);
```

gibt ein Fragezeichen am Bildschirm aus. Überall dort, wo das Zeichen ein Wort in Klammern ist, stellt es ein Zeichen dar, das als solches nicht am Bildschirm ausgedruckt werden kann, obwohl es die Bildschirmanzeige beeinflussen könnte (zum Beispiel die Bewegung des Cursors).

Sind Sie mit dem ASCII-Code bereits vertraut, werden Sie bemerken, daß die ersten 32 Zeichen der Liste graphische Symbole enthalten, die bei GW-BASIC anstelle der herkömmlichen Steuerzeichen verwendet werden.

Beim Editieren eines Programms können Sie am Bildschirm ein Zeichen ausgeben, für das sich auf Ihrer Tastatur keine Taste befindet, indem Sie den Code bestehend aus drei Ziffern durch Betätigen der Alt-Taste ausgeben. Auf diese Weise ist es möglich, Zeichen, die sich nicht auf der Tastatur befinden, in eine Zeichenketten-Konstante einzubeziehen.

Die Werte, die in der Spalte &H der Liste aufgeführt sind, bedeuten die hexadezimalen Äquivalente der Dezimalcodes.

ASCII-Wert	&H	Zeichen	Steuerzeichen	ASCII-Wert	&H	Zeichen
000	00	(Null)	NUL	032	20	(Leerzeichen)
001	01	☺	SOH	033	21	!
002	02	●	STX	034	22	"
003	03	♥	ETX	035	23	#
004	04	♦	EOT	036	24	$
005	05	♣	ENQ	037	25	%
006	06	♠	ACK	038	26	&
007	07	(Lautsprecher)	BEL	039	27	'
008	08	■	BS	040	28	(
009	09	(Tab)	HT	041	29	)
010	0A	(Zeilenvorschub)	LF	042	2A	*
011	0B	(Grundstellung)	VT	043	2B	+
012	0C	(Papiervorschub)	FF	044	2C	,
013	0D	(CR)	CR	045	2D	-
014	0E	♫	SO	046	2E	.
015	0F	☼	SI	047	2F	/
016	10	►	DLE	048	30	0
017	11	◄	DC1	049	31	1
018	12	↕	DC2	050	32	2
019	13	!!	DC3	051	33	3
020	14	¶	DC4	052	34	4
021	15	§	NAK	053	35	5
022	16	▬	SYN	054	36	6
023	17	↨	ETB	055	37	7
024	18	↑	CAN	056	38	8
025	19	↓	EM	057	39	9
026	1A	→	SUB	058	3A	:
027	1B	←	ESC	059	3B	;
028	1C	(Cursor nach rechts)	FS	060	3C	<
029	1D	(Cursor nach links)	GS	061	3D	=
030	1E	(Cursor nach oben)	RS	062	3E	>
031	1F	(Cursor nach unten)	US	063	3F	?

ASCII-Wert	&H	Zeichen	ASCII-Wert	&H	Zeichen
64	40	@	096	60	
65	41	A	097	61	a
66	42	B	098	62	b
67	43	C	099	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[	123	7B	{
92	5C	\	124	7C	¦
93	5D	^	125	7D	}
94	5E	↑	126	7E	~
95	5F	—	127	7F	⌂

ASCII-Wert	&H	Zeichen	ASCII-Wert	&H	Zeichen
128	80	Ç	160	A0	á
129	81	ü	161	A1	í
130	82	é	162	A2	ó
131	83	â	163	A3	ú
132	84	ä	164	A4	ñ
133	85	à	165	A5	Ñ
134	86	å	166	A6	ª
135	87	ç	167	A7	º
136	88	ê	168	A8	¿
137	89	ë	169	A9	⌐
138	8A	è	170	AA	¬
139	8B	ï	171	AB	½
140	8C	î	172	AC	¼
141	8D	ì	173	AD	¡
142	8E	Ä	174	AE	«
143	8F	Å	175	AF	»
144	90	É	176	B0	▒
145	91	æ	177	B1	▒
146	92	Æ	178	B2	▓
147	93	ô	179	B3	│
148	94	ö	180	B4	┤
149	95	ò	181	B5	╡
150	96	û	182	B6	╢
151	97	ù	183	B7	╖
152	98	ÿ	184	B8	╕
153	99	Ö	185	B9	╣
154	9A	Ü	186	BA	║
155	9B	¢	187	BB	╗
156	9C	£	188	BC	╝
157	9D	¥	189	BD	╜
158	9E	Pt	190	BE	╛
159	9F	ƒ	191	BF	┐

ASCII-Wert	&H	Zeichen	ASCII-Wert	&H	Zeichen
192	C0	└	224	E0	α
193	C1	┴	225	E1	ß
194	C2	┬	226	E2	Γ
195	C3	├	227	E3	π
196	C4	─	228	E4	Σ
197	C5	┼	229	E5	σ
198	C6	╞	230	E6	µ
199	C7	╟	231	E7	τ
200	C8	╚	232	E8	Φ
201	C9	╔	233	E9	Θ
202	CA	╩	234	EA	Ω
203	CB	╦	235	EB	δ
204	CC	╠	236	EC	∞
205	CD	═	237	ED	φ
206	CE	╬	238	EE	ε
207	CF	╧	239	EF	∩
208	D0	╨	240	F0	≡
209	D1	╤	241	F1	±
210	D2	╥	242	F2	≥
211	D3	╙	243	F3	≤
212	D4	╘	244	F4	⌠
213	D5	╒	245	F5	⌡
214	D6	╓	246	F6	÷
215	D7	╫	247	F7	≈
216	D8	╪	248	F8	°
217	D9	┘	249	F9	∙
218	DA	┌	250	FA	·
219	DB	▉	251	FB	√
220	DC	▄	252	FC	n
221	DD	▌	253	FD	2
222	DE	▐	254	FE	■
223	DF	▀	255	FF	(blank 'FF')

Eine Anzahl von durch INKEY$ gelesenen Codes sind Zwei-Code-Zeichen, die nicht zum ASCII-Code gehören. Wenn INKEY$ eines dieser Sonderzeichen liest, ist das erste Zeichen ein Nullzeichen (Code 000). In einem solchen Fall sollte Ihr Programm das zweite Zeichen der Zeichenkette untersuchen, die von INKEY$ zurückgegeben wurde. Dieses Zeichen ist gewöhnlich der Tastencode, der sich auf die jeweilige Position auf der Tastatur bezieht. Dies gilt nur dann, wenn der Modus der betreffenden Taste (Shift, Control, Alt oder keiner) in der folgenden Liste aufgeführt ist.

Zweites Zeichen	**Taste(n)**
3	(Nullzeichen) NUL

15	(shift tab) ←
16-25	Alt- Q,W,E,R,T,Y,U,I,O,P
30-38	Alt- A,S,D,F,G,H,J,K,L
44-50	Alt- Z,X,C,V,B,N,M
59-68	Funktionstasten F1 bis F10 (wenn nicht im programmierten Zustand)
71	Home
72	Cursor, nach oben
73	Pg Up
75	Cursor nach links
77	Cursor nach rechts
79	Ende
80	Cursor, nach unten
81	Pg Dn
82	Ins
83	Del
84-93	F11-F20 (Shift-F1 bis F10)
94-103	F21-F30 (Ctrl- F1 bis F10)
104-113	F31-F40 (Alt-F1 bis F10)
115	Ctrl-Cursor, nach links (vorausgehendes Wort)
116	Ctrl-Cursor, nach rechts (nächstes Wort)
117	Ctrl-End
118	Ctrl-Pg Dn 119Ctrl-Home
120-131	Alt- 1,2,3,4,5,6,7,8,9,0,–,=
132	Ctrl-Pg Up

Fehlermeldungen

Hat GW-BASIC in Ihrem Programm einen Fehler entdeckt, wird die Ausführung des Programms gewöhnlich abgebrochen und eine Fehlermeldung, die Sie darüber informiert, was Sie falsch gemacht haben, wird am Bildschirm ausgegeben. Ein Fehlerzustand ist meistens darauf zurückzuführen, daß Ihr Programm von GW-BASIC verlangt hat, die Regeln der Sprache zu verletzen, jedoch auch eine Unstimmigkeit zwischen GW-BASIC und einem Peripheriegerät kann manchmal die Ursache sein.

Wenn Sie es wünschen, können Sie Fehlerzustände mit Hilfe des ON ERROR-Befehls und den beiden GW-BASIC-Variablen ERR und ERL aufdecken. In diesem Fall sollte Ihre Fehlerbehandlungsroutine etwas unternehmen, um den Fehlerzustand zu beseitigen, falls Sie beabsichtigen, Ihr Programm weiterlaufen zu lassen, als wäre nichts geschehen. Wird von der Fehlerbehandlungsroutine nichts unternommen, kommt GW-BASIC auf Sie mit der gleichen Fehlermeldung zurück oder das Ergebnis Ihres Programms wird unzuverlässig.

Der erste Abschnitt dieses Anhangs besteht aus einem Überblick über die Fehlernummern. Wenn Sie in Betracht ziehen, was für Fehlermöglichkeiten in Ihrem Programm auftreten können, ziehen Sie einfach die untenstehende Liste zu Rate. Um eine noch ausführlichere Beschreibung Ihrer Fehlernummer zu erhalten, werfen Sie einen Blick in den zweiten Abschnitt. Beachten Sie, daß Sie auch selbst Fehlerzustände definieren und ebenso Fehlernummern zuteilen können (siehe ERROR).

Der zweite Abschnitt behandelt die GW-BASIC Fehlermeldungen in alphabetischer Reihenfolge und informiert Sie über die mögliche Ursache des Fehlers.

ÜBERBLICK ÜBER FEHLERZAHLEN

Fehlernummer	Fehlermeldung
1	NEXT without FOR

2	Syntax error
3	RETURN without GOSUB
4	Out of data
5	Illegal function call
6	Overflow
7	Out of memory
8	Undefined line number
9	Subscript out of range
10	Duplicate Definition
11	Division by zero (Fehlerbehandlungsroutine nicht möglich)
12	Illegal direct
13	Type mismatch
14	Out of string space
15	String too long
16	String formula too complex
17	Can't continue
18	Undefined user function
19	No RESUME
20	RESUME without error
22	Missing operand
23	Line buffer overflow
24	Device Timeout
25	Device Fault
26	FOR without NEXT
27	Out of paper
29	WHILE without WEND
30	WEND without WHILE
50	FIELD overflow
51	Internal error
52	Bad file number
53	File not found
54	Bad file mode
55	File already open
57	Device I/O error
58	File already exists
61	Disk full
62	Input past end
63	Bad record number
64	Bad file name
66	Direct statement in file
67	Too many files
68	Device unavailable
69	Communication buffer overflow

70	Disk Write Protect
71	Disk nor ready
72	Disk media error
74	Rename across disk
75	Path/file access error
76	Path not found
--	Unprintable error

Fehlermeldungen **Fehlernummer**

Bad file mode **54**

Unzulässiger Dateimodus. Sie haben versucht, PUT oder GET bei einer Datei mit sequentiellem Zugriff oder bei einer geschlossenen Datei zu verwenden, um eine Datei mit direktem Zugriff zu laden oder einen OPEN-Befehl ohne zulässigen Dateimodus (Input, Output, Append oder Random) auszuführen.

Ein derartiger Fehler kann ebenfalls vorkommen, wenn Sie versuchen, aus einer für die Ausgabe eröffneten Datei zu lesen oder eine Nicht-ASCII-Datei einzufügen. Versuchen Sie den OPEN-Befehl in Ihrem Programm zu überprüfen.

Bad file name **64**

Unzulässiger Dateiname. Eine unzulässige Form wurde für die Dateinamenvergebung mit dem einem KILL, NAME oder FILES-Befehl (d.h ein Dateiname mit zu vielen Zeichen).

Bad file number **52**

Unzulässige Dateinummer. Ein Befehl verweist auf eine Datei mit einer Dateinummer, die nicht eröffnet wurde oder sich außerhalb der Dateinummern befindet, die beim Laden von GW-BASIC angegeben wurden oder der Name des Geräts war zu lang oder unzulässig.

Überprüfen Sie den Namen und die Nummer der Datei im OPEN-Befehl für die betreffende Datei.

Bad record number **63**

Unzulässige Datensatznummer. In einer PUT- oder GET-Anweisung war die Datensatznummer entweder größer als die höchstzulässige (16,777,215) oder gleich Null.

Can't continue **17**

Fortsetzung des Programmablaufs nicht möglich. Sie versuchten, den CONT-Befehl anzuwenden, um ein Programm weiterlaufen zu lassen, das:

1. aufgrund eines Fehlers angehalten wurde.

2. nach einer Programmunterbrechung abgeändert wurde

3. oder überhaupt nicht vorhanden ist.

Versichern Sie sich, daß das Programm geladen ist und beginnen Sie es mit dem RUN-Befehl.

Communication buffer overflow 69
Überlauf des Übertragungspuffers. Kommt dann vor, wenn eine Eingabeanweisung für die Datenübertragung ausgeführt wird und der Eingabepuffer bereits voll ist. Um die Eingabe noch einmal zu versuchen, verwenden Sie eine Fehlerbehandlungsroutine (ON ERROR). Wenn die Zeichen schneller empfangen werden als das Programm sie verarbeiten kann, sind folgende Maßnahmen zu berücksichtigen:

1. Vergrößern Sie über die /C Option den Datenübertragungs -empfangsbuffer beim Laden von GW-BASIC.

2. Wenn der Sender ein „handshaking" Protokoll unterstützen kann, ziehen Sie eines in Ihr Datenübertragungsprogramm ein. Somit kann das verarbeitende Programm an der Eingabe Schritt halten.

3. Verwenden eine niedrigere Übertragungsgeschwindigkeit für das Senden und Empfangen.

Device Fault 25
Gerätefehler. Ein Interface-Adapter gibt einen Hardwarefehler zurück.

Bei der Übertragung von Daten auf eine Übertragungsdatei wird angezeigt, daß eines oder mehrere der geprüften Signale (festgelegt beim OPEN „COM...-Befehl) in der angegebenen Zeit nicht gefunden wurde.

Device I/O Error 57
Geräte-Ein-/Ausgabe-Fehler. Diese Fehlermeldung wird angezeigt, wenn es zu einem Überlauf-, Paritäts- oder Rahmenfehler bei der Datenübertragung kommt. Wenn die Zeichenlänge 7 oder weniger Datenbit beträgt, wird das höchstwertigste Bit im fehlerhaften Byte gesetzt.

Device Timeout 24
Geräte-Zeitsperre. Diese Fehlermeldung erscheint am Bildschirm, wenn eines oder mehrere der vom OPEN „COM-Befehl geprüfte Signale innerhalb der angegebenen Zeit nicht gefunden wurden.

Sie können eine Fehlerbehandlungsroutine anweisen, die Operation (RESUME) noch einmal zu versuchen, Sie sollten jedoch die Zahl der Versuche begrenzen, ansonsten könnte Ihr Programm in einer Schleife unendlich ablaufen.

Device Unavailable **68**
Einheit nicht vorhanden. Sie versuchten, eine Datei in einer nicht vorhandenen Einheit zu eröffnen. Vielleicht ist keine Hardware vorhanden bzw. Ihr Programm hat die Datenübertragung zum Gerät unterbrochen. Zum Beispiel wurde die COM1-Anweisung durch Angabe der /C Option beim Wert 0 während des Ladevorgangs von GW-BASIC unterbrochen. Ist dies der Fall, müssen Sie wieder auf die Betriebsebene (SYSTEM) zurückkehren und GW-BASIC erneut laden.

Direct statement in file **66**
Befehl im Direktmodus auf einer Datei. Beim Laden (LOAD) oder Verketten (CHAIN) einer Datei im ASCII-Format ist man auf einen Befehl im Direktmodus gestoßen. Der Befehl LOAD oder CHAIN wird beendet. Die ASCII-Datei sollte nur aus GW-BASIC-Befehlen mit Zeilennummern bestehen. Der Fehler wurde von einem Zeilenvorschubzeichen bei der Eingabe verursacht.

Disk full **61**
Platte voll. Sämtlicher Speicherplatz ist voll belegt. Sobald GW-BASIC auf diesen Fehlerzustand stößt, schließt GW-Basic sämtliche Dateien. Löschen Sie alle unnötigen Dateien oder verwenden Sie eine neue Platte. Versuchen Sie, erneut mit der Platte zu arbeiten oder lassen Sie das Programm nochmals vom Anfang an ablaufen.

Disk Media Error **72**
Physischer Plattenfehler. Tritt dann ein, wenn die Steuereinheit einen Hardwarefehler am Datenträger entdeckt. Es handelt sich dann um eine beschädigte Platte.
Kopieren Sie sämtliche vorhandene Dateien auf eine neue Platte und formatieren Sie die beschädigte Platte erneut.

Disk not Ready **71**
Diskette nicht in Bereitschaft. Die Diskette ist wahrscheinlich nicht richtig eingelegt.

Disk Write Protect **70**
Platten-Schreibschutz wirksam. Tritt dann ein, wenn Sie versuchen, auf eine schreibgeschützte Platte zu schreiben. Die Fehlermeldung kann ebenfalls von einem Hardware-Fehler stammen.

Überprüfen Sie, ob Sie die richtige Platte verwenden. Entfernen Sie dann den Schreibschutz und und versuchen Sie mit der Platte zu arbeiten.

Division by zero **11**

Division durch Null. Man ist in einem Ausdruck auf eine Division durch Null oder auf eine negative Potenzierung einer Null gestoßen. Ist die Ursache der Fehlermeldung eine Division durch Null, erscheint der Unendlichwert und die Nummer, die die Fehlermeldung verursachen, am Bildschirm. Eine falsche Potenzierung ergibt einen positiven Unendlichwert. Dieser Fehler ist nicht auffindbar.

Duplicate Definition **10**

Doppelte Definition. Zwei DIM-Befehle werden für die gleiche Feldvariable erteilt; ein DIM-Befehl wird erteilt, nachdem die Standardgröße 10 für die betreffende Feldvariable durch eine implizite Definition eingerichtet wurde; oder GW-BASIC ist auf einen OPTION BASE-Befehl nach der ersten Benennung oder nach dem Gebrauch einer Feldvariablen gestoßen.

FIELD overflow **50**

Feldüberlauf. Ein FIELD-Befehl, mit dem man versucht hat, mehr Byte als für die Datensatzlänge einer Datei mit direktem Zugriff zulässig ist in einem OPEN-Befehl zuzuordnen; oder man ist auf das Ende des FIELD-Pufferspeichers bei sequentieller Datenübertragung (PRINT#. WRITE#, INPUT#) auf eine Datei mit direktem Zugriff gestoßen.

Prüfen Sie, ob der OPEN- dem FIELD-Befehl entspricht. Bei sequentieller Datenübertragung auf eine Datei mit direktem Zugriff, darf die Länge der gelesenen bzw. geschriebenen Daten die Datensatzlänge der Datei mit direktem Zugriff nicht überschreiten.

File already exists **58**

Datei bereits vorhanden. Der in einem NAME-Befehl festgelegte Dateiname ist mit dem auf einer Platte bereits verwendeten Dateinamen identisch.

Erteilen Sie den NAME-Befehl mit einem anderen Namen.

File already open **55**

Datei bereits eröffnet. Ein OPEN-Befehl mit sequentiellem Ausgangsmodus wurde für eine bereits eröffnete Datei erteilt; oder
ein KILL-Befehl wurde für eine bereits eröffnete Datei erteilt.
Prüfen Sie, ob Sie nur dann einen OPEN-Befehl einer Datei erteilen, wenn Sie sequentiell auf Sie schreiben oder Sie mit einem Anhang versehen. Bevor Sie den KILL-Befehl erteilen, schließen Sie die Datei ab.

File not found **53**

Datei nicht gefunden. Ein LOAD-, KILL-, NAME-, oder OPEN-Befehl befindet sich nicht auf der aktivierten Diskette.

Prüfen, ob sich die richtige Diskette im angegebenen Laufwerk befindet, und ob der vollständige Dateiname, falls erforderlich einschließlich Pfad, richtig eingegeben wurde. Dann Betrieb wieder aufnehmen.

FOR without NEXT **26**

FOR ohne entsprechendes NEXT. Man ist auf einen FOR-Befehl ohne das entsprechende NEXT gestoßen. Vielleicht läuft gerade eine FOR-Schleife ab, wenn das physische Ende des Programms erreicht wurde.

Ihr Programm sollte einen NEXT-Befehl beinhalten.

Illegal direct **12**

Unzulässiger Direktbefehl. Sie haben versucht, einen Befehl, der im Direktmodus ungültig ist, als Befehl im Direktmodus (z.B. DEF FN) einzugeben.

Geben Sie den Befehl als Teil einer Programmzeile ein.

Illegal function call **5**

Unzulässiger Funktionsaufruf. Ein Parameter, der sich außerhalb des zulässigen Bereichs befindet, wird einer Systemfunktion übertragen. Diese Fehlermeldung kann ebenfalls verursacht werden durch:

1. Einen negativen oder unwahrscheinlichen Indexwert für eine Feldvariable.

2. Ein unzulässig angegebenes Argument (negativ bzw. Null) für eine numerische Funktion.

3. Einen Aufruf einer USR-Funktion, bei der die Anfangsadresse noch nicht mit dem DEF USR-Befehl definiert wurde.

4. Ein falsches Argument für einen Zeichenketten-Verarbeitungsbefehl oder eine Funktion.

5. Eine negative Datensatznummer, die mit Hilfe der Befehle GET oder PUT verwendet wird.

6. Einen Versuch ein geschütztes BASIC Programm aufzulisten oder zu editieren.

7. Einen Versuch Zeilennummern zu löschen, die nicht existieren.

Input past end **62**

Eingabe über das Ende hinaus. Ein INPUT-Befehl wird ausgeführt, nachdem alle Daten in die Datei eingegeben wurden (INPUT) bzw. INPUT wurde bei einer leeren Datei versucht.

Zur Vermeidung dieses Fehlers ist die EOF-Funktion zu verwenden, um das Ende der Datei zu ermitteln. Die Fehlermeldung wird ebenfalls ausgegeben, wenn versucht wird, aus einer Datei zu lesen, die für eine Ausgabe oder Erstellung eines Anhang eröffnet wurde.

Wenn Sie aus einer sequentiellen Ausgangs- oder Anhangsdatei lesen wollen, müssen Sie die betreffende Datei zunächst schließen und dann wieder für die Eingabe öffnen.

Internal error **51**

Interner Fehler. Eine interne Störung ist in GW-BASIC aufgetreten. Kopieren Sie nochmals Ihre Diskette. Überprüfen Sie die Hardware und versuchen Sie nochmals die Operation.

Line buffer overflow **23**

Überlauf des Zeilenpuffers. Sie haben die Eingabe einer Zeile mit zu vielen Zeichen versucht.

Trennen Sie die Mehrfachbefehle so, daß sie sich auf mehr als einer Zeile befinden; oder Sie können auch Zeichenkettenvariablen anstelle von Konstanten verwenden.

Missing operand **22**

Fehlender Operand. Ein Ausdruck enthält einen Operator, auf dem kein Operand folgt. Überprüfen Sie, ob alle erforderlichen Operanden im Ausdruck enthalten sind.

NEXT without FOR **1**

NEXT ohne FOR. Eine Variable in einem NEXT-Befehl entspricht keiner zuvor ausgeführten unpaarigen Variablen mit einem FOR-Befehl. Das Programm ist so anzuanpassen, daß NEXT wieder ein paariges FOR besitzt.

No Resume **19**

Kein RESUME. Eine Fehlerbehandlungsroutine wurde ohne RESUME Befehl eingegeben. Prüfen Sie, ob Sie in Ihrer Fehlerbehandlungsroutine RESUME einbezogen haben, damit ein weiterer Programmablauf möglich ist. Es ist möglich, einen ON ERROR GOTO 0-Befehl Ihrer Fehlerbehandlungsroutine hinzuzufügen, so daß BASIC eine Meldung für jeden nicht erfaßten Fehler ausdruckt.

Out of data **4**

Keine Daten (DATA) mehr vorhanden. Eine READ-Anweisung wurde ausgeführt, als keine Daten (DATA) mehr vorhanden waren.

Korrigieren Sie Ihr Programm so, daß sich genügend Datenwörter in den DATA-Listen für alle READ-Befehle im Programm befinden.

Out of memory **7**

Speicherplatz nicht ausreichend. Ein Programm ist zu groß oder weist zu viele FOR-Schleifen oder GOSUB-Routinen auf, zu viele Variable oder zu komplizierte Ausdrücke. Eine umfangreiche Verwendung des PAINT-Befehls mit besonders zackigen Ecken kann auch diese Fehlerbedingung schaffen.

Es ist möglich den CLEAR-Befehl am Anfang Ihres Programms zu verwenden, um mehr Stapelplatz bzw. Speicherplatz zu reservieren.

Out of paper **27**

Papier nicht ausreichend. Papier fehlt oder der Drucker ist nicht eingeschaltet. Legen Sie Papier in den Drucker ein und prüfen Sie, ob der Drucker angeschlossen und eingeschaltet ist; erst dann setzen Sie Ihr Programm fort.

Out of string space **14**

Nicht genügend Platz für Zeichenketten. Zeichenketten-Variable haben GW-BASDIC veranlaßt, den frei gebliebenen Speicherplatz zu überschreiten. BASIC wird den Zeichenketten solange dynamisch Speicherplatz zuweisen, bis kein Platz mehr vorhanden ist.

Overflow **6**

Überlauf. Das Ergebnis einer Berechnung ist zu lang, daß es von GW-BASIC ausgedruckt werden kann. Ganzzahlüberlauf stoppt den Programmablauf. Andernfalls erhält man den „Unendlichwert" mit dem entsprechenden Vorzeichen als Ergebnis und der Programmablauf wird fortgesetzt. Der Ganzahlüberlauf ist die einzige Überlaufart, die einer Fehlerbehandlungsroutine unterzogen werden kann. Um einen Ganzzahlüberlauf zu korrigieren, müssen Sie kleinere Zahlen oder Variable von einfacher bzw. doppelter Genauigkeit verwenden.

Wenn ein numerisches Ergebnis so klein ist, daß es GW-BASIC nicht ausdrucken kann, ist das Ergebnis Null und der Programmablauf wird ohne Fehlermeldung fortgesetzt.

Path/file access error **75**

Zugriff auf Pfad/Datei verweigert. Bei der versuchten Ausfürung eines OPEN, MKDIR, CHDIR oder RMDIR-Befehls konnte NCR-DOS keine

richtige Verbinduzng zwischen Pfad und Dateinamen herstellen, zum Beispiel haben Sie versucht, ein Inhaltsverzeichnis zu eröffnen oder das aktuelle Inhaltsverzeichnis zu löschen bzw. einen Datenträgerkennamen zu eröffnen. Andernfalls haben Sie versucht, eine Nur-Lese-Datei zu eröffnen.

Path not found **76**
Pfad nicht gefunden. Bei einer versuchten Ausführung eines OPEN, MKDIR, CHDIR oder RMDIR-Befehls konnte NCR-Dos den angegebenen Pfad nicht finden.

Rename across disks **74**
Es wurde versucht, eine Datei mit einer neuen Laufwerkbezeichnung umzubenennen. Dies ist nicht zulässig.

Resume without error **20**
Resume ohne Fehlermeldung. Ein RESUME-Befehl wurde außerhalb einer Fehlerbehandlungsroutine entdeckt. Die häufigste Ursache dafür ist die, daß das Programm über den normalen Bereich hinaus bis zu den Fehlerbehandlungsroutinen weiterläuft. Um dies zu verhindern ist ein STOP oder END-Befehl überall dort anzuwenden, wo angenommen wird, daß das Programm anhält.

RETURN without GOSUB **3**
RETURN ohne GOSUB. GW-BASIC hat einen RETURN-Befehl entdeckt, für den es noch keinen entsprechenden GOSUB-Befehl gibt. Die häufige Ursache für diese Fehlermeldung ist die, daß Ihr Programm weiter bis zu den Subroutinen gelaufen ist. Um dies zu verhindern, ist ein STOP oder ein END-Befehl überall dort zu verwenden, wo der Programmablauf gestoppt werden soll.

String formnula too complex **16**
Zeichenkettenformel zu komplex. Ein Zeichenkettenausdruck ist zu lang und zu komplex. Der Ausdruck sollte in kleinere Ausdrücke aufgeteilt werden.
Diese Fehlermeldung kann ferner bei einer Zusammensetzung von mehr als 10 Pfadbefehlen (CHDIR, MKDIR, RMDIR) auftreten. Mit dem Befehl PRINT "" wird dieser Fehlerzustand behoben.

String too long **15**
Zeichenkette zu lang. Es wurde versucht, eine Zeichenkette von 255 Zeichen einzurichten. Eine Verwendung kleinerer Zeichenketten wird empfohlen.

Subscript out of range **9**
Indexwert außerhalb des Bereichs. Ein Feldvariablenelement wird

durch einen Indexwert, der sich außerhalb der Feldvariablen-Dimensionen oder unter den falschen Nummern der Indexwerte befindet, angesprochen. Siehe ebenso „Unzulässiger Funktionsaufruf". Die Verwendung der Feldvariablen ist zu prüfen. Es ist möglich, daß einer Variablen ein Indexwert zugeordnet wurde, die sich nicht in der Feldvariablen befindet.

Syntax error **2**

Syntaxfehler. GW-BASIC hat eine Zeile entdeckt, die eine falsche Zeichenfolge enthält (z.B. unpaarige Klammern, falsch buchstabierte Befehle, falsche Satzzeichen etc.). Eine solche Fehlermeldung kann auch dann auftreten, wenn man mit einem READ-Befehl versucht, Daten (DATA) mit falscher Benennung einer Variablen zuzuordnen.

Too many files **67**

Zu viele Dateien. Mit Hilfe des SAVE- oder OPEN-Befehls wurde versucht, eine neue Datei einzurichten, obwohl bereits alle Einträge im Inhaltsverzeichnis voll sind, bzw. die Dateibezeichnung ist ungültig.

Type mismatch **13**

Typenvermischung. Mit Ihrem Programm haben Sie versucht, einer numerischen Variablen oder umgekehrt einen Zeichenkettenwert zuzuweisen, oder ein Argument vom falschen Typ wurde einer Funktion übergeben.

Undefined line number **8**

Nicht definierte Zeilennummer. Ein Befehl bezieht sich auf eine nicht vorhandene Zeilennummer. Eine vorhandene Zeilennummer (z.B. eine REM-Zeile) ist zu verwenden.

Undefined user function **18**

Nicht definierte Anwender-Funktion. Sie haben eine Funktion aufgerufen, bevor Sie die Funktion definiert haben (DEF FN). Prüfen, ob das Programm die DEF FN-Anweisung ausführt, bevor die Funktion angewendet wird.

Unprintable error **–**

Fehler ohne Meldung. Für den aufgetretenen Fehler ist keine Fehlermeldung möglich. Dies ist normalerweise der Fall bei einem ERROR-Befehl mit einem nicht definierten Code. Vergewissern Sie sich, daß Sie mit allen Fehlercodes, die Sie erstellen, umgehen können, vorausgesetzt Sie wollen, daß Ihr Programm weiter abläuft ohne Eingriff seitens des Programmierers.

Wend without WHILE **30**

WEND ohne WHILE. Das Programm ist auf einen WEND-Befehl ohne dazugehörigem WHILE gestoßen.

WHILE without WEND **29**

WHILE ohne WEND. Ein WHILE-Befehl hat kein entsprechendes WEND. WHILE war immer noch aktiv, als das physikalische Ende des Programms erreicht wurde.

Das Programm ist so zu berichtigen, daß jedes WHILE ein entsprechendes WEND besitzt.

Zusätzliche Funktionen

Dieser Anhang enthält mathematische Funktionen in der GW-BASIC Sprache, die nicht Bestandteil von GW-BASIC sind Eine zweckmäßige Art solche vom Anwender definierten Funktionen zu speichern und aufzurufen, ist die Verwendung des DEF FN-Befehls und der FN-Funktion. Um zum Beispiel eine Funktion, die den Cotangens eines Wertes ergibt, ist folgender Befehl zu verwenden:

```
DEF FN COTAN(X) = 1/TAN(X)
```

Um die definierte Funktion aufzurufen, wird folgender Befehl benötigt:

```
ERGEBNIS = FNCOTAN(WINKEL>
```

Beachten Sie, daß Sie die internen GW-BASIC Funktionen als Bestandteil Ihrer Funktionsdefinitionen verwenden können. In diesem Beispiel wird die Funktion TAN verwendet. Innerhalb der Vorschriften für die Namensvergebung von Variablen können Sie Ihrer Funktion jeden gewünschten Namen vergeben. Zum Beispiel würde die folgende Funktionsdefintion dem gleichen Zweck dienen:

```
DEF FN SUNTAN(LOTION) = 1/TAN(LOTION)
```

und könnte mit:

```
ENIGMA = SUNTAN(SOMETHIN)
```

aufgerufen werden. Jedoch später könnten Sie sich wahrscheinlich nicht mehr erinnern, warum Sie ursprünglich die Funktion definiert haben. Daher ist es sinnvoll, bedeutungsvolle Funktionsnamen zu verwenden.

Logarithmus zur Basis B	LOGB(X) = LOG(X)/LOG(B)
Sekante	SEC(X) = 1/COS(X)
Kosekante	CSC(X) = 1/SIN(X)
Kotangente	COT(X) = 1/TAN(X)
Invertierter Sinus	ARCSIN(X) = ATN(X/SQR(1-X*X))
Invertierter Kosinus	ARCCOS(X) = 1.570796 -ATN(X/SQR(1-X*X))

Invertierte Sekante	ARCSEC(X) = ATN(SQR(X*X-1)) +(X<0*3.141593
Invertierte Kosekante	ARCCSC(X) = ATN(1/SQR(X*X-1)) +(X<0)*3.141593
Invertierte Kotangens	ARCCOT(X) = 1.57096-ATN(X)
Hyperbolischer Sinus	SINH(X) = (EXP(X)-EXP(–X))/2
Hyperbolischer Kosinus	COSH(X) = (EXP(X)+EXP(–X))/2
Hyperbolischer Tangens	TANH(X) = (EXP(X)-EXP(–X)) /(EXP(X)+EXP(–X))
Hyperbolische Sekante	SECH(X) = 2/(EXP(X)+EXP(–X))
Hyperbolische Kosekante	CSCH(X) = 2/(EXP(X)-EXP(–X))
Hyperbolischer Kotangens	COTH(X) = (EXP(X)+EXP(–X)) /(EXP(X)-EXP(–X))
Invertierter hyperbolischer Sinus	ARCSINH(X) = LOG(X+SQR(X*X+1))
Invertierter hyperbolischer Kosinus	ARCCOSH(X) = LOG(X+SQR(X**X-1))
Invertierter hyperbolischer Tangens	ARCTANH(X) = LOG(1+X)/(1-X))/2
Invertierte hyperbolische Sekante	ARCSECH(X)= LOG(1+SQR(1-X*X))/X)
Invertierte Kosekante	ARCCSCH(X) = LOG((1+SGN(X) *SQR(1+X*X))/X)
Invertierter hyperbolischer Kotangens	ARCCOTH(X) = LOG((X+1))/(X-1))/2

Dezimale und Hexadezimale Zahlen

Die Umwandlung einer Dezimalzahl in ihr hexadezimales Äquivalent erfolgt bei GW-BASIC in Form der HEX$-Funktion (siehe Kapitel 4 „Befehle und Funktionen"). Diese Funktion ergibt das hexadezimale Äquivalent für die Dezimalzahl im Bereich – 32768 bis 65535 (falls die Zahl negativ ist, wird ein Zweierkompliment verwendet).

Hier ein Beispiel für die Verwendung der HEX$-Funktion:

```
PRINT HEX$(255)
ergibt
FF
```

GW-BASIC selbst liefert keine Funktion zur Umwandlung von Hexadezimalzahlen in Dezimalzahlen, jedoch Sie könnten das folgende Programm verwenden, bei dem eine Hexadezimalzahl in einen positiven Dezimalwert umgewandelt wird. Geben Sie eine beliebige Hexadezimalzahl ein und verwenden Sie die Großbuchstaben A bis F. Stellen Sie der Zahl kein &H voraus.

```
9900 INPUT "Hex Zahl";H$
9905 DEC=0
9910 FOR C%= 1 TO LEN(H$)
9920 CH$=MID$(H$,C%,1)
9930 IF (CH$<"0" OR CH$>"9") AND (CH$<"A" OR CH$<"F")
     THEN GOTO 9900
9940 DEC=16*DEC-(CH$<"A")*(ASC(CH$)-48)-
     (CH$>"9")*(ASC(CH$)-55)
9950 NEXT C%
9960 PRINT "Hex ";H$;" = ";DEC;" dezimal"
```

Außerhalb des normalen Programmierbereichs sind hohe Zeilennummern zu verwenden. Sie können dann dieses Programm im Speicher lassen oder an das gerade bearbeitete Programm anhängen (MERGE). Somit ist eine schnelle Umwandlung von hexadezimalen in dezimale Zahlen anhand des direkten Befehls GOTO 9900 beim Programmieren möglich.

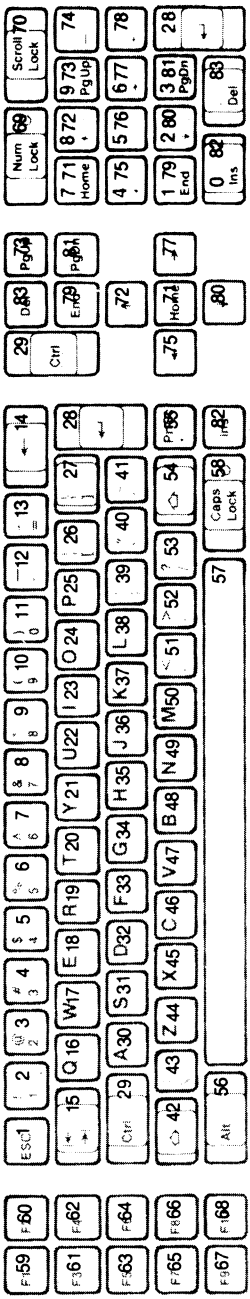
Positionen auf der Tastatur

Ihr NCR-PC bezieht sich intern auf die jeweiligen Tasten der Tastatur anhand der Tastenpositionen. Normalerweise beziehen Sie sich beim Programmieren mit GW-BASIC auf die jeweilige Taste anhand des auf der Taste aufgedruckten Namens, zum Beispiel:

```
10 K$ = INKEY$ : IF K$ = "N" THEN GO TO 10
```

Es gibt jedoch zwei Umstände, bei denen die Kenntnis der Tastenposition unerläßlich ist, erstens beim Lesen eines Zwei-Code-Zeichens (siehe Anhang B) von der Tastatur und beim Abfangen einer bestimmten Tastenbetätigung (siehe KEY).

In der Zeichnung auf der nächsten Seite befindet sich die jeweilige Positionsnummer oben in der linken Ecke einer jeden Taste der Tastatur Ihres NCR-PC.


F1 59
F2 60
F3 61
F4 62
F5 63
F6 64
F7 65
F8 66
F9 67
F10 68
ESC 1
2
3
4
5
6
7
8
9
10
11
12
13
← 14
Tab 15
Q 16
W 17
E 18
R 19
T 20
Y 21
U 22
I 23
O 24
P 25
26
27
↵ 28
Ctrl 29
A 30
S 31
D 32
F 33
G 34
H 35
J 36
K 37
L 38
39
40
41
Shift 42
43
Z 44
X 45
C 46
V 47
B 48
N 49
M 50
51
52
53
Shift 54
Prt 55
Alt 56
57
Caps Lock 58
F11 82
F12 83
Num Lock 69
Scroll Lock 70
7 71 Home
8 72 ↑
9 73 PgUp
4 75 ←
5 76
6 77 →
1 79 End
2 80 ↓
3 81 PgDn
0 82 Ins
. 83 Del
Ctrl
PgUp 73
PgDn 81
↑ 72
Home 71
← 75
End 79
↓ 80
→ 77

NCR
Personal Computer Division
Augsburg, Germany

017-0040227