GeoBasic

Programming Manual

The Official BASIC Programming Language for GEOS C-64/128 Users

Developed by Berkeley Softworks
Distributed by *RUN* Magazine

geoBASIC

The BASIC programming language for GEOS C-64/128 users.

IDG Communications/Peterborough, Inc. (Publishers of *RUN* Magazine), has licensed geoBASIC from Berkeley Softworks.

Customer service and technical support questions should be referred to *RUN* Magazine, 80 Elm St., Peterborough, NH 03458. Technical support is also provided on the *RUN* geoBASIC area on QuantumLink.

geoBASIC runs in C-64 mode (40 columns) on C-64/128.

JUNE 1990

Dear geoBASIC owner,

Congratulations and thank you for purchasing geoBASIC.

You are the proud owner of a powerful version of BASIC that gives GEOS users the flexibility to develop their own programs in BASIC. With this product (in C-64 40-column mode) you have everything you need to begin developing your own BASIC applications in GEOS, taking advantage of the features that have made GEOS so easy to use—pull-down menus, dialog boxes, icons and mouse pointer for easy point-and-click operations. You'll now be able to create programs that use icons, menus, sprites and dialog boxes.

We're pleased to be able to present geoBASIC, and plan to support this product both through the magazine and in the *RUN* geoBASIC area on QuantumLink. We encourage users to upload the programs they have developed to the Q-Link area and also to submit their work to *RUN* for possible publication. We also encourage users to access the Q-Link area for information on the latest geoBASIC developments, programming information, forums, software to download, sample applications and answers to your questions and comments.

We were encouraged by the GEOS community to bring this product to market. We have worked diligently to provide a top-quality and much-needed product to the marketplace. We feel that we have succeeded, and welcome your comments regarding our efforts.

In putting together this manual, we made several assumptions. One is that you are familiar with GEOS and know how to load and open GEOS files.

Also, it is not the intent of this manual to teach you how to

program. If you are interested in using geoBASIC, then you should already be familiar with BASIC programming. If you've never programmed before, then we strongly suggest that you develop some BASIC programming skills either through a course or a book before you use this program. The purpose of this manual is to teach you how to use geoBASIC and its commands.

I would like to extend my appreciation to the following:

First, to the gang at Berkeley Softworks for developing this program and licensing it to *RUN*. Through their cooperation, we are able to bring this product to market.

Second, to project manager Lou Wallace, who was the driving force for this product here at *RUN*. Without his persistence, long hours under impossible deadlines, foresight and dedication to this project, geoBASIC would not now be available to the GEOS community.

Third, to the group of tireless testers and reviewers of this product, particularly Joe Buckley and Bill Coleman. These programming wizards were responsible for testing and developing applications. Their keen insights and knowledge of GEOS contributed to the success of this product.

This manual contains all the information you need to get under way developing your own programs in GEOS. All you need besides this manual is a little imagination. Good luck and happy programming.

Dennis Brisson
Editor-In-Chief
RUN

TABLE OF CONTENTS

CHAPTER ONE—AN OVERVIEW OF geoBASIC.....1

Features of geoBASIC 2
The geoBASIC Utilities 2
Using This Manual 4

CHAPTER TWO—INTRODUCTION TO geoBASIC PROGRAMMING.....5

Elements of a geoBASIC Program.....5
The geoBasic Screens.....6
Running geoBASIC.....6
The Text Editor.....7
Loading a Program 7
Saving a Program 8
Listing a Program 8
Editing Keystrokes 8

CHAPTER THREE—A SIMPLE geoBASIC TUTORIAL.....10

Good Programming Practices.....10
The Sample Application.....13

CHAPTER FOUR—PROGRAMS, DATA AND VARIABLES 14

The geoBASIC Editor 14
Editing Keys.....15
Clearing the Screen and Listing Your Program.....16
The geoBASIC Menus 16
Elements of the geoBASIC Language 20
Constants 21
Variables 23
Arrays 25

Expressions....27
Arithmetic Expressions....27
Hierarchy of Arithmetic Operators....31
String Expressions....32

CHAPTER FIVE—THE geoBASIC COMMAND REFERENCE....33

Elements....33
geoBASIC Commands....34

CHAPTER SIX—DISK AND FILE PROGRAMMING....102

Disk and File Commands....102
VLIR File Description....107

CHAPTER SEVEN—THE geoBASIC UTILITY PROGRAMS....109

The Menu Editor....109
The Bitmap Utility....112
Dialog Box Editor....115
Icon List Utility....120
Sprite Editor....122
The Editor Screen....127

CHAPTER EIGHT—THE geoBASIC DEBUGGER....131

Entering Expressions or Breakpoints....131
Debug Modes....134
Errors....134

CHAPTER NINE—geoBASIC ERROR MESSAGES....136

Chapter One—An Overview of geoBASIC

BASIC is a language for programming your computer. It is the most easily learned and most widely used of the computer languages available. Thousands of ordinary people have learned BASIC and successfully program their computers. You don't need to have an educational background with years of college and a penchant for all-night sessions to be a programmer in BASIC. In fact, probably more people create programs in BASIC for the C-64 than any other computer. And this process is made even easier by GEOS.

GEOS is the Graphics Environment Operating System from Berkeley Softworks. It includes intuitive tools such as pull-down menus, information boxes that provide choices right on the screen, on-screen pictures (called icons) and the mouse pointer for easy point-and-click operations. An example of how these elements combine to make operations easy should illustrate what we mean. When you insert a disk in the drive, all you need to do to find out what files are on the disk is move the mouse pointer up to the menu at the top of the screen labeled "DISK" and click the left button. The submenus will become visible, and you would then click on "OPEN". The various files on the disk become visible in a "window" on the screen. Each file has a picture associated with it which gives further information about the file. To print out a file, click once on the icon of the text file you want to print. This will cause it to become highlighted. Click it a second time and you will have picked up an outline of the icon. As the mouse pointer moves, an outline of the file will move with it. Move the icon to the printer icon, and press the left button. When you release the left button, the printer will start up and out will come the printed copy. If the printer is not hooked up, then a box (called a dialog box) will appear on the screen informing you of this. When you move the mouse pointer to the button labeled "OK" and click the left button, the box will disappear, and you will be right back where you started.

FEATURES OF GEOBASIC

geoBASIC is a full-featured BASIC for the C-64. It includes a text editor for entering and editing programs, as well as menus for special features. With geoBASIC, you can use icons, menus, sprites and dialog boxes in your own programs, to make them look professional as well as easy to use. Color and sound, text windows and drawing commands are available, as is full support for the mouse. Structured loops, subroutines, mathematical functions and access to machine language subroutines are all supported. You even have special commands that provide access to disk files for storing and recovering data.

THE GEOBASIC UTILITIES

Included with geoBASIC are five utilities that you can use to build various items to use in your programs. Each of the utilities works similarly. They are:

Menu Editor

The Menu Editor enables you to build menus for use in your programs. The words which appear at the top of the screen are submenus. Moving the mouse pointer up to a menu and clicking the left button makes the items in that menu "drop down," appearing on the screen under the submenu. To select an item, move the mouse pointer to that item and click the left button. If you change your mind about using an item under a submenu, simply move the mouse pointer away from the dropped-down items without clicking the left button, and the submenu will close up again.

The Menu Editor is an interactive utility. As you specify each submenu, it appears at the top of the screen as part of the menu. You can specify the number of submenus in the menu and the exact text of each submenu, and these can be changed at any time. You can also specify where the program will branch when the user selects (clicks on) a particular item.

Bitmap Editor

Bitmaps are small pictures that you can place on the screen (see Icon Editor, below) or in a dialog box (see Dialog Box Editor, below). The Bitmap Editor lets you draw and save bitmaps for later use. You can specify the size of the final bitmap and use the mouse pointer to turn points off and on in an enlarged drawing area. As you proceed the editor also shows you what the final bitmap looks like in actual size.

Icon Editor

The Icon Editor lets you use bitmaps defined with the Bitmap Editor in your programs. Once a bitmap has been drawn and saved using the Bitmap Editor, the Icon Editor lets you define the coordinates on the screen where you want the bitmap to appear when you use the geoBASIC command ICON.

Dialog Box Editor

Dialog boxes are boxes that appear on the screen in response to the geoBASIC DIALOG command. Their purpose is to present information to the user and get a choice or acknowledgement from the user. After the user has made a selection from a dialog box, the box is automatically removed from the screen and whatever was hidden by the box is restored on the screen.

A dialog box can contain text, expressions (which can change depending on the value of program variables), icons you have designed using the Icon Editor, and system icons, such as the OK, Cancel and Disk buttons. The Dialog Box Editor lets you specify what objects you want to appear in the dialog box, what coordinates (location) you want for each object, and the text or expression to use for those objects that require a text message or expression. When you select a type of object to use, you will be prompted on the screen for all the information required to draw that object in the dialog box. All of the specified quantities can be edited at any time using the mouse pointer.

Sprite Editor

Sprites (also called Movable Object Blocks) are used for animating sections of the screen. Often used for games, they can also be used in many other types of applications. The Sprite Editor is a major part of the geoBASIC utility set, offering unheard of power in the generation and use of sprites. For details, see the Utilities chapter later in this manual.

USING THIS MANUAL

The rest of the chapters in this manual are designed to get you comfortable with geoBASIC as well as to provide a complete reference to geoBASIC and the included utilities.

Several chapters act as an introduction and hands-on tutorial for geoBASIC, while others contain a complete listing of all geoBASIC commands, often including samples using those commands in a program segment. Also included is a description of all the various special elements of the GEOS environment (such as dialog boxes, sprites, bitmaps, etc.)

WHAT YOU SHOULD KNOW

Certain things are not within the scope of this manual to teach, such as how to set up your equipment. If you are unsure how to do this, refer to the manual that came with your Commodore. This manual also will not teach you how to program, how to become a programmer or how to use GEOS. It assumes that you have already programmed with another language and are familiar with the basic tenets and practices of programming a computer.

Some technical aspects of the Commodore 64/128 are covered in the Commodore 64 Programmer's Reference Guide. The details of the SID (sound) chip, and how to manipulate sprites are covered there, and are not repeated here. The goal of this manual is to familiarize you with geoBASIC and explain how to use the commands present in this language.

Chapter 2—Introduction to geoBASIC Programming

ELEMENTS OF A GEOBASIC PROGRAM

A program is made up of lines of commands. All of the valid commands and details of how to use each are detailed in the reference section of this manual. At the beginning of each line there must be a line number. The line number not only identifies the beginning of a line of commands, but also determines the order of execution of the lines of commands. Consider the following:

```
10 ...COMMANDS...
20 ...COMMANDS...
```

This represents two lines of a program, numbered 10 and 20. The "...COMMANDS..." represents valid geoBASIC commands that would normally follow the line number. If you typed in a line beginning with the number 15 followed by commands, you would have the following if you listed the program using the command LIST:

```
10 ...COMMANDS...
15 ...COMMANDS...
20 ...COMMANDS...
```

Notice how the line numbered 15 ended up between lines 10 and 20. It is a good idea, when first typing lines into your program, to number them in increments of 10, so you have plenty of room to insert extra lines later.

Line numbers are also important because certain commands make reference to a particular line, and one way to identify the line in question is by using its line number (see GOSUB and GOTO in the reference section). Another way to identify a line is by using a label. A label must follow a line number on the line. The label name must be preceded by an @ symbol, and can be up to six characters long, provided the label name begins with a letter and does not contain any valid geoBASIC commands in the label

name. The label name can also be followed by a colon (:) and another command. Examples of a label are:

20 @LBL: ...COMMANDS...
30 @LBL

The screen on your C-64 is 40 characters wide. Each line of commands in a geoBASIC program can be up to 240 characters, which would occupy up to six screen lines. So, one line of the program may take up more than one line on the screen. It is important to distinguish between these. A command can be on two lines of the screen, but must be fully contained in one line of the program.

Each line in the program must contain at least one command or a space. If you type in a line number with no commands or spaces after it, that line will not show up in your program. Each line may have more than one command if you choose. Separate multiple commands with colons (:)

10 @LBL: FOR N=1 TO 10:PRINT N:NEXT N

THE GEOBASIC SCREENS

geoBASIC maintains two screens for you to work on. The first is the text screen, where you type in your program and edit it. The other screen is the graphics screen, where your program actually runs. Because the two screens are separate, your program listing remains intact while the program runs, and any results produced by the program remain on the graphic screen, even after you return to the text screen to further edit your program. If you want, during editing you can switch back and forth between the graphics screen and the editor screen by pressing the F7 key.

RUNNING GEOBASIC

To begin using geoBASIC from the GEOS desktop screen, insert the disk containing geoBASIC into your disk drive. Move the mouse pointer to the disk submenu and click on the open item

to show the contents of the disk. Move the mouse pointer to the icon marked "GEOBASIC" and double click the left button. geoBASIC will load and its initial screen will appear.

THE TEXT EDITOR

geoBASIC includes a powerful text editor for entering and editing your programs. A complete listing of its capabilities is included in the reference section.

LOADING A PROGRAM

To load a new program, you must use the dialog box that appears when you begin using geoBASIC. Choose from the three options: Create New Document, Open Existing Document, and Quit To DeskTop.

If you are going to create a new document, click on the Create button and another box will appear asking you to "Please enter a new filename:". Type in the filename you want to use and press [RETURN]. If you want to access a different disk drive, click on the Drive button. If you wish to go back to the original dialog box, click on the button marked Cancel. The filename you type in must not exist on the disk or you will receive a message that states "File exists, choose another". If you receive this message, click on the OK button to return to the previous dialog box.

If you are going to open an existing document, click on the Open button. This will bring up a dialog box that lists the names of the first fifteen geoBASIC files on the disk. You may select a file in the list and click on Open to begin working on it, change drives using the Drive button, or return to the previous dialog box by clicking on Cancel. If there are too many files on the disk for them all to be visible in the dialog box, click on the up or down arrows which will appear to scroll through the list of files.

Note: If you are editing a file with geoBASIC and want to work with another file, you must close the current file using the

Close item in the File submenu. This will return you to the initial geoBASIC dialog box so you can load another file.

SAVING A PROGRAM

When you have revised a program in memory, it is important to save the changes to disk or they will be lost when you quit geoBASIC. To save the changes to disk, select the Update item from the File submenu. Saving also occurs automatically when you use the Close item under the File submenu to stop working with the current file.

LISTING A PROGRAM

To look at your program on the screen, you must list it. If you want to see the whole program, you may type in the LIST command (with no line number in front of it) or select the list item under the Edit submenu. If the program is too long to fit on the screen, the lines at the top of the screen will scroll off the screen as new lines are listed at the bottom. To pause the scrolling, use [F5] as a start/stop LIST toggle. Press it once to stop a listing, then press it again to continue listing the program. You may also abort the LIST command (or the menu item equivalent) by pressing the [RUN/STOP] key. To list any line number of the program, type the LIST command followed by the line number you wish to view. You may also list a range of line numbers by using the LIST command followed by the starting line number, a comma, and the ending line number. For more information on the use of the LIST command, see the reference section.

EDITING KEYSTROKES

To change anything in your program, simply LIST the line you wish to modify on the screen (see above). The flashing box you see is known as the cursor, and whatever you type will be placed on the screen at the cursor's position. You may type in anything you like from the keyboard. To get upper case letters, press the [Shift] key while pressing the letter key. The [Shift] key also gives

you access to the characters above the number keys. Once you have finished typing in a line of BASIC text, press the [RETURN] key to enter it. You may move the cursor by using the mouse pointer or the arrow keys. To use the mouse pointer, move the pointer to the spot where you want the cursor to appear and click the left button. The arrow key with the up and down arrows on it will move the cursor down the screen if you press just the key, and the cursor will move up the screen if you hold down the [Shift] key at the same time as you press the up/down arrow key. The arrow key with the left and right arrows on it will move the cursor to the right if you press just the key, and it will move the cursor to the left if you hold down the [Shift] key at the same time as you press the left/right arrow key.

Chapter 3—A Simple geoBASIC Tutorial

Included on your geoBASIC disk is a sample application called "Sample Appl." In this section of the manual, we will load and look at this application. It illustrates many of the commands in geoBASIC and uses most of the utilities as well. It also illustrates some good programming practices. If you don't have geoBASIC running right now, insert the disk containing geoBASIC into your disk drive. Move the mouse pointer to the disk submenu and click on the open item to show the contents of the disk. Move the mouse pointer to the icon marked "GEOBASIC" and double click the left button. geoBASIC will load and its initial screen will appear. Click on the Open button, select "Sample Appl" and click on Open. After a few moments, you should be looking at the "Ready" prompt. Or, you can just click on the "Sample Appl" icon directly, which will cause geoBASIC to load first, followed by the application. To verify that the program has actually been loaded, type LIST and press [RETURN]. If you have a printer hooked up, you may wish to print out a copy of this program, as it is rather long. To do so, move the pointer up to the File submenu and press the left button. The menu items should become visible. Move the pointer to the print item and press the left button. The program listing should print out on your printer.

GOOD PROGRAMMING PRACTICES

Before moving on to look at our Sample Application, we'll take a look at some good programming practices which are illustrated by the application. While it is not strictly necessary to follow these practices, programming will be much easier, and it will be easier to make changes to the program in the future if these suggestions are followed.

1. Use indenting to make your code more readable.

FOR loops, IF clauses and subroutines are much easier to read and understand if you indent them. This is especially true for

nested FOR loops and IF clauses that are on more than one screen line. As an example, check out the Sample Application listing. The indenting makes it much easier to read.

Example of Nested FOR loops:

```
10 FOR X=1 TO 100
20     PRINT X
30     PRINT X+5
40     FOR Y=1 TO 10
50         PRINT Y
60         PRINT Y+2
70     NEXT Y
80 NEXT X
```

Example of Multi-line IF statement:

```
10  IF RENAMED=0 THEN NEWFILE$="HELLO THERE":RENAMED=1
```

Example of labelled subroutine:

```
10 GOSUB @STARTDRAWING
20 REM
30 END
40 @STARTDRAWING
50 XCLICK=MOUSEX(0)
60 YCLICK=MOUSEY(0)
70 XOLD=XCLICK
80 YOLD=YCLICK
90 RETURN
```

2. Use labels as much as possible.

Labels make your code more easily understood. Also, subroutines and lines accessed by the utilities (Dialog, Sprite, etc.) should be labeled so that they can be accessed properly by the utilities. You may use up to 127 labels within a single program. Keep in mind that geoBASIC's RENUMBER command does not renumber GOTOs or GOSUBs, so you must use labels if you expect to renumber your program. As an example, look at the Sample Application. Liberal use of labels with long, descriptive names

makes it a lot easier to figure out what is going on. For example, GOSUB @Handleclick is much clearer than GOSUB 2680! To further increase the clarity of the listing, try to use just a label on a line.

```
10 @HandleClick
```

NOTE: If you need to change the line number of a label, first delete the old line, THEN type in the new line number with the label.

3. Put blank lines between subroutines.

This helps separate parts of the program. To have a blank line (nothing except a line number on it), select Insert Mode and type the line number and a space, then press [RETURN].

4. Use mixed case for variable names.

Again, this makes your code easier to read. Some examples are nmFlags, cardFlag, Done, etc.

5. Avoid multiple statements on a line.

This cannot always be avoided, but your code will be easier to read if you don't try to crowd too many statements on a line. With IF statements (which must be on the same logical line), you can take advantage of the fact that a logical line can take up to six screen lines. Thus, you can press [RETURN] after each statement in the IF statement, moving each part of the IF statement onto a different screen line. Indenting will also make the IF statement easier to read.

```
10 IF RENAMED THEN
NEWFILE$ = RENAME$:
RENAMED = FALSE
```

Notice the colon separating each command. This is required for proper syntax.

THE SAMPLE APPLICATION

The sample application listing is pretty self-explanatory and should be easy to follow, not only because it follows the suggestions above, but uses plenty of REM statements to explain what each section does. This is also good programming practice, and you should study the techniques used, especially the MAINLOOP command, which is the heart of the application. The program stays in this command, branching only when a menu or icon is selected, then it branches to the appropriate subroutine. After the subroutine is executed, execution returns to MAINLOOP, to await the next selection.

Now we'll add a short segment to the sample application to let you see how it works. Type in the following:

```
6670 DBSTRN "ARE YOU SURE?",ANS$
6672 IF ANS$<>"Y" AND ANS$<>"Y"
THEN RETURN
```

Enter these lines, and press [RETURN] at the end of each one. To enter line 6672, type in the first part (first line), then use the space key to move the cursor to the second line and type it in. Entering a program is just this simple!

To save the changed version of the program, move the mouse pointer up to the File submenu and click on the Update item.

Chapter 4—Programs, Data and Variables

THE GEOBASIC EDITOR

Text and Entry Modes

The first screen you see when you start geoBASIC is the editor text screen. To enter text into the body of your geoBASIC program, you must type that text into the geoBASIC editor. The flashing box is known as the cursor, and whatever you type will be placed on the screen at the cursor's position. You may type in anything you like from the keyboard. To get upper case letters, press the [Shift] key while pressing the letter key. The [Shift] key also gives you access to the characters above the number keys. Once you have finished typing in a line of BASIC text, press the [RETURN] key to enter it.

There are two modes of text entry: overstrike and insert. To switch back and forth between overstrike and insert mode, hold down the [Shift] key and press the INST/DEL key on the top right corner of your keyboard. The normal mode of text entry is overstrike. Whatever you type will replace what is on the screen at the cursor position. In the insert mode, whatever you type will still appear at the cursor, but anything located to the right of the cursor on the same line will be pushed to the right to make room for the new text. Any text which no longer fits on the line it is on will be pushed onto the next line to make room. The maximum size of one line of geoBASIC commands is 240 characters (6 lines). If you extend a line longer than 6 screen lines, part of it will be lost, most likely resulting in a syntax error. When you are in insert mode, the cursor will become a solid block to indicate the change in mode.

MOVING AROUND THE KEYBOARD

To move the cursor around the text on the screen, you can use the arrow keys in the lower right corner of the keyboard or your

mouse pointer. The arrow key with the up and down arrows on it will move the cursor down the screen if you press just the key, and the cursor will move up the screen if you hold down the [Shift] key at the same time as you press the up/down arrow key. The arrow key with the left and right arrows on it will move the cursor to the right if you press just the key, and it will move the cursor to the left if you hold down the [Shift] key at the same time as you press the left/right arrow key. You may also move the cursor anywhere on the screen by moving the mouse pointer where you want the cursor to appear and pressing the left mouse button.

To move the cursor to the top right corner of the screen, press the [Home] key.

EDITING KEYS

To edit your text, you can use the [DEL] key at the top right corner of your keyboard or the <- key at the top left corner. Their functions are similar, but not identical. The [DEL] key erases the character immediately to the left of the cursor and moves the character under the cursor and everything to the right of the cursor one space to the left. The <- key erases the character directly under the cursor and moves any characters to the right of the cursor one space to the left. To erase the entire line that the cursor is on, hold down the Commodore key and press [DEL]. NOTE: Never delete an entire program line by backspacing, deleting or overstriking the characters in the line and pressing [RETURN].

The editor includes a tab key function. To use it, press the [CONTROL] and [I] keys together. There are tab stops at line positions 8, 12, 16, 20, 24, 28, 32, and 36. The tab key function can be very handy for indenting portions of your program to make it easier to read. If you activate the tab function by pressing [CONTROL] [I] while in overstrike mode, the cursor will move to the new position. If you are in insert mode the cursor will move to the new tab position and the number of spaces that the cursor moved will be inserted in the line at the old cursor position. You also have access to powerful editing functions from the menus —

see the menu section for a description of these.

CLEARING THE SCREEN AND LISTING YOUR PROGRAM

You can clear the screen, removing everything from it, by holding down the [Shift] key and pressing the [HOME/CLR] key. To get a listing of your program on the screen so you can edit it, use the LIST command (see below) or the list item under the Edit submenu. To change any line in the program, simply move the cursor onto that line, type in your changes, and press [RETURN] to enter the changes .

THE GEOBASIC MENUS

The geoBASIC submenus located at the top of the screen and the items under each submenu control many of the editing and file handling functions. To select an item, move the mouse pointer up to the submenu which contains the item and click the left button. The submenu will open up, showing the items associated with it. Then move the mouse pointer to the item you want and click the left button again. The submenu will close up and your choice will be acted upon. The submenu items are listed below, in the order they appear under their respective submenus.

GEOS

geoBASIC Info

This item displays a Dialog box on the screen which shows the version number of the geoBASIC, the author's name, and the copyright information.

FILE

CLOSE

This item saves your program to disk and closes the file,

returning you to the initial dialog box which appears when you first run geoBASIC. At that point, you may load and use another geoBASIC file.

UPDATE

This item saves a copy of your current file to disk. Although this is done automatically when you leave geoBASIC (see Close above), it is a good idea to save a copy of your changes periodically so that they won't be lost in the event of a power failure or if you accidently turn off the computer without using Close.

RENAME

This item renames your file on disk and in memory. The current file will be saved to disk using the name specified here whenever you subsequently use update or close. Selecting this item brings up a dialog box for you to Please enter a new filename:. Type in the new filename you want and press [RETURN]. To change your mind, just press [RETURN] without entering a filename or click on the Cancel button. If the filename you selected already exists on the disk, you will end up with two files with the same name. The contents of the file on the disk will replace the current file, and the current file will be lost. Exercise caution when using this function!

PRINT

This item prints a complete listing of your program to your printer. Make sure your printer is hooked up and turned on before choosing this item.

QUIT

This item first saves the current file to disk, then returns to the GEOS deskTop.

EDIT

LIST

This item lists the current program on the screen. If the

program is too long to appear on one screen, the lines of the program will scroll off the top of the screen as new lines are listed at the bottom. Use the F7 key to start and stop the listing process. To break into the listing, press the [RUN/STOP] key.

SPRCOL

This menu item allows you to choose the two sprite colors which remain the same for each sprite. The third sprite color (which can be different for each sprite) is chosen using the Sprite utility. Selecting this item brings up a dialog box for you to Click on the boxes to choose the sprite multicolors. There are two lines, labeled Multicolor 1 and Multicolor 2. Alongside each of these lines is a small, colored box. Each time you click on one of the colored boxes, it changes color, cycling through the sixteen available sprite colors. When you are satisfied with the color selection, click on the OK button.

OPTIONS

RUN

This item runs your program. It has the same effect as typing RUN in the text window.

RENUMBER

This item will renumber your program for you. This is useful if you need to insert new line numbers between existing line numbers, but no intermediate line numbers are available. Selecting this item brings up a dialog box which requests you to Enter amount to renumber. Type in the number and press [RETURN] to proceed with the renumbering or click on Cancel to cancel the renumbering. The number you enter sets both the first line number and the increment between line numbers. For example, if you enter 100, the first line will be line 100, then lines 200, 300, 400 and so on. Remember, RENUMBER ignores the line numbers following GOTOs and GOSUBs, so use labels for subroutines.

RESIZE

Resize changes the heap size for your program.

MAKE APPL

This turns a geoBASIC program into a standalone executable file. Once created, this standalone file is not editable.

UTILITIES

MENU

This item activates the Menu Editor, which enables you to construct your own menu for use in your programs. For a complete description of the Menu Editor, see the chapter on Utilities.

DIALOG

This item activates the Dialog Box Editor, which enables you to construct your own dialog boxes for use in your programs. For a complete description of the Dialog Box Editor, see the chapter on Utilities.

ICON

This item activates the Icon List Editor, which enables you to construct your own icon lists for use in your programs. For a complete description of the Icon List Editor, see the chapter on Utilities.

BITMAP

This item activates the Bitmap Editor, which enables you to design bitmaps for use with the Icon List Editor or the Dialog Box Editor. For a complete description of the Bitmap Editor, see the chapter on Utilities.

SPRITE

This item activates the Sprite Editor, which enables you to design sprites for use in your program. For a complete description of the Sprite Editor, see the chapter on Utilities.

ELEMENTS OF THE GEOBASIC LANGUAGE

Line Numbers and Labels

Each line in a geoBASIC program must begin with a line number. This line number not only identifies the line, but also sets the order in which lines will normally be executed in the program. For example:

```
10 PRINT "HELLO THERE"
20 FOR X=1 TO 10
30 NEXT X
```

Each of these lines begins with a number. The lines are listed (and executed) in the order of the line numbers. By numbering the lines every 10, you can easily insert other lines in between the lines you have already. For example, you could add the line: 25 PRINT X, in which case the listing above would look like:

```
10 PRINT "HELLO THERE"
20 FOR X=1 TO 10
25 PRINT X
30 NEXT X
```

Notice how line 25 was inserted between lines 20 and 30. The line numbers are also used as the target of GOSUB and GOTO statements (see the explanation of these commands in the reference section). See also the RENUMBER menu item.

Labels provide a way to give a name to a line. While the line number is still necessary on a line with a label, that line can then be referenced by the label when using GOSUB or GOTO.

Labels must be the first command of the line, immediately after the line number itself. There may be spaces between the line number and the label. The label must be preceded by the "@" sign and only the first six letters of the label name are significant. The label names are case-sensitive, that is, upper and lower case letters are not the same, i.e. @START is a different label than @Start. A label and a variable (see below) may have the same name but are

considered different by the program even if they do. Labels take on the value of the line they are on, and can be used in mathematical formulas just like variables, except that you may not try to set the value of a label. The value of the label is set when the label is declared by placing it at the beginning of a line in the program. When using a label in a formula, make sure to use the leading "@" to distinguish it as a label.

Labels may not be declared (placed at the beginning of a line to establish their value) more than once. If you place the same label at the beginning of two different lines, a LABEL REDEFINED error will occur. It is very important that you always delete the first occurrence of a label before changing its location. And keep in mind that the total number of labels in a single program cannot exceed 127.

For example:

```
10 @START: PRINT "HELLO"
20 FOR X=1 TO 10
30 PRINT SIN(X)
40 NEXT X
50 GOTO @START
```

In line 10, the label START is declared. geoBASIC can tell START is a label because of the "@" in front of the label name. The label START actually takes on the value 10. Thus, in line 50, the statement GOTO @START really is saying GOTO 10.

CONSTANTS

Constants are values which do not change during the execution of your geoBASIC program. There are two types of numeric constants, which are just numbers. Integer constants are whole numbers (numbers with no decimal point) which can range in value from -32768 to 32767. You may not use a comma to separate the digits in an integer constant, and leading zeros are ignored. Each integer constant uses two bytes of memory. Some examples

of integer constants are:

-10
1256
0

A Floating Point constant is a positive or negative number which can contain decimal points and fractional portions. Once again, commas are not allowed between the numbers. Floating point constants can be represented two ways. The first way is as a simple number with up to 9 digits. The number can range from –999999999 to 999999999. If you specify more than nine digits, the number will be rounded based on the tenth digit. If the tenth digit is greater than or equal to 5, then the number will be rounded upward. If the tenth digit is less than 5, the number will be rounded down. Some examples of floating point constants represented as simple numbers are:

12.45
3.1415924
66666.66
.01

If the number is less than .01 or greater than 999999999, then the floating point constant will be printed in scientific notation. A number printed in scientific notation looks something like:

1.23456E07.

The first part of the number is the digits to the left of the "E". This part is called the "mantissa" and is a simple floating point number, with the decimal point to the right of the first digit. The letter "E" lets you know that you are viewing the number in exponential form. The numbers following the "E" are called the "exponent", and they are the integer power of 10 that the mantissa should be multiplied by to get the actual number. The decimal point in the mantissa would be moved the number of decimal places to the right indicated by the exponent if the exponent is positive. If the exponent is negative, then the decimal point would be moved the number of places to the left indicated by the exponent. Thus, 3E3 would be 3000 because it is 3 multiplied by

10 to the 3rd power (1000). Both the mantissa and the exponent can be negative. If the mantissa is negative, then the whole number is less than zero. If the exponent is negative, then the number is between 0 and 1 if the mantissa is positive and between 0 and -1 if the mantissa is negative. Thus, 3E-3 is .003 because 10 to the -3 power is .001. Even in scientific notation there is a limit to the range of numbers you can handle. The largest number is 1.70141183E+38. If your expression results in an answer which is larger, you will get an ?OVERFLOW ERROR. The smallest number (closest to zero) is 2.93873588E-39. If a smaller number is the result of your expression, the answer will be given as zero with no error message. Some examples of floating point scientific notation numbers are:

1.3456E07
3.4E-5
-7.7E-09

String constants are groups of alphanumeric text such as letters, numbers, and symbols. When you enter a string constant from the keyboard, it may be up to the length of one full line (240 characters less what is taken up by the line number and any other statements on the line). A string constant can contain blanks, letters, numbers, and punctuation in any combination. The string constant must be enclosed in double quotes (") and thus may not contain any double quotes in the string. You may leave off the double quotes at the end of the string if the string is the last item on a line or is separated from the next item by a colon (:). Some examples of string constants are:

"HELLO"
"BERKELEY SOFTWORKS , BERKELEY"
"$56,000"

VARIABLES

Variables are data used in your geoBASIC program which can change as the program executes. Variables are identified by their names. A variable name can be any length but only the first three

characters are considered significant: a variable is identified by the first three letters of its name, which must be unique. Variable names are case-sensitive, so AbC is a different variable than aBc. Any alphanumerical character or number can be used in a variable name, but the first character must be a letter. Also, you may not use a geoBASIC keyword (command) in a variable. If you accidently include a keyword in a variable name, you will get a ?SYNTAX ERROR.

Values can be assigned to a variable by setting it equal to a constant, another variable, or an expression. If you assign a value to a variable which already has a value, the former value will be lost and the variable takes on the new value. Variables have the same types as the constants discussed above: integers, floating point numbers and strings. The last character in the name chosen for the variable sets the type of variable. If the "$" is the last character in the variable name, then it is a string variable. If the "%" is the last character in the variable name, then it is an integer variable. If neither of these characters is used as the last character in the variable name, then the variable is a floating point variable.

A floating point number may be equated to an integer variable (the fractional part will be ignored) and an integer number may be equated to a floating point variable. However, you may not equate a string variable to a number nor a numeric variable to a string. Attempting to do so will cause an error.

Examples:

D$="HELLO THERE"	**(string variable equal to a string constant)**
A$="HELLO"+" THERE"	**(string variable equal to a string expression)**
B$=D$+" "+A$	**(another string expression)**
D=1.234	**(floating point variable equal to a constant)**
D=4.5+(6*7)/C	**(and equal to an expression)**
D%=6	**(integer variable equal to a constant)**
D%=(6*7)+5	**(and an expression)**

ARRAYS

An array is a table of data items which are identified by a single variable name. This name must follow the same rules as other variable names. Within the array, the values (known as elements) are identified by an element number. The variable name identifies which table of data is being referred to, while the element number specifies exactly which item in the table is to be used. Take for example an array "A". To refer to the third element in A, you would use the statement:

```
A(3)
```

To set the value of the third element in array "A" you would use a statement like:

```
A(3)=10.5
```

Array elements can be used anywhere that normal variables can, and the element number can even be the result of an expression:

```
A(3+5)=11
```

or even another array element:

```
A(B(4))=5
```

Array names can be string, integer or floating point variables. Their types are identified the same way as regular variables — with the endings $ (string), % (integer), or no ending (floating point array). All elements of the array must match the array name type.

Arrays may have up to 255 dimensions. In the example above, the array "A" has only one dimension. You can visualize an array with two dimensions as a table where the first dimension is the rows and the second dimension is the columns:

	0	1	2	3
0	1.1	1.2	3.4	5.6
1	2.6	7.9	3.6	7.9
2	3.5	6.8	1.9	0.6
3	4.6	6.8	2.4	9.7

This table is a two dimensional array. If the array name is ARR, then the first element is ARR(0,0), and it is equal to 1.1. Arrays are very useful. In the example above, there are 16 different values stored in the single array name "ARR". If you couldn't use arrays, you would have to come up with 16 different variables. An array such as DNP(10,10,10) has over 1000 elements. You couldn't possibly come up with that many variables, and accessing each variable would be very difficult and require tremendous amounts of BASIC code. However, to access any element of this array, all you would need to do is use a statement such as:

X=5:Y=6:Z=3:DNP(X,Y,Z)=23

As you can see, the element number (also referred to as a subscript) must be enclosed in parentheses following the array name with the subscript for each dimension separated from the others by commas. To use any array with more than 10 elements, you must use the DIM statement (see the reference section) to declare how many elements there will be in each dimension:

DIM DNP(10,10,10)

This DIMension statement actually declares 11 elements in each dimension, 0 through 10. The total number of elements in this array is thus 11*11*11, or 1331. If you try to use an element number outside the range declared in the DIM statement, you will get a ?BAD SUBSCRIPT error. Array elements are automatically filled with zeroes (for floating point or integers) or nulls (for string arrays) when created.

The memory usage for arrays is as follows:

5 bytes for the array name
2 bytes for each dimension of the array
2 bytes per element of the array for integers
5 bytes per element for floating points
3 bytes per element for strings

plus 1 byte per character for each element in a string array.

Examples:

A$(0)="HELLO WORLD"	**(string array)**
AA%(BB%)=5	**(integer array)**
AA(4*X+6,5,7)=A*B	**(3 dimensional floating point array)**
QR%(AA%(B,G))=4	**(nested integer arrays)**

EXPRESSIONS

Expressions are formulas and equations formed using constants, variables, arrays, labels and operators. The operators can be arithmetic, logical, or relational. This combination of items is designed to produce a result which can be used in the program. There are two types of expressions: arithmetic and string. Arithmetic expressions will be covered first.

ARITHMETIC EXPRESSIONS

An arithmetic expression uses arithmetic operators, constants, labels and variables to produce a result which is an integer or floating point number. If a label is used, it takes on the numeric value of the line which it was declared on:

```
10 @LBL:A=0
```

In this example, the label @LBL takes on the value 10, and can be used in an expression. An arithmetic expression is normally broken into data items called operands, separated by operators. One or more operators, combined with one or more operands, normally form an expression. The arithmetic operators which may be used in an expression are:

- \+ Addition
- \- Subtraction
- * Multiplication
- / Division
- ^ Exponentiation (raising to a power)

Relational operators are normally used to compare the values

of two operands, but they can also produce a numeric result and so can be used in arithmetic expressions. If the relation tested is true, then the result is -1, and if the relation being tested is false, then the result is 0. The relational operators are:

< Less than
= Equal to
> Greater than
<= Less than or equal to
>= Greater than or equal to
<> Not equal to

Examples:

2=3-1 result is true (-1)
20+4>5+9 result is false (0)

The logical operators (AND, OR, NOT) can be used to modify the results of using relational operators or to produce an arithmetic result. With the AND operator, each bit in the first expression is ANDed against the corresponding bit in the second expression. The bit in the result is equal to 1 if the bit in each expression is 1. If the bit in either expression is 0, then the bit in the result is zero. Thus, you get:

0 AND 0 = 0 1 AND 0 = 0 0 AND 1 = 0 1 AND 1 = 1

Example:

X=32007 AND 28761 (result is 28673)

With the OR operator, each bit in the first expression is ORed against the corresponding bit in the second expression. The bit in the result is equal to 1 if the bit in either expression is 1. If the bit in both expressions is 0, then the bit in the result is zero . Thus, you get:

0 OR 0 = 0 1 OR 0 = 1 0 OR 1 = 1 1 OR 1 = 1

Example:

X=32007 OR 28761 (result is 32095)

The third logical operator is NOT. NOT turns a 1 into a 0 and a 0 into a 1:

NOT 0 = 1 NOT 1 = 0

For all the logical operators, the numbers operated on must evaluate to between 32767 and –32768.

In addition to bitwise operations, the logical operators can be combined to modify the results of comparisons. For the AND operator, the result evaluates as true only if both of the expressions are true, and if either expression is false, then the result is false. The "truth table" for the AND statement looks like:

First Expression	Second Expression	Result Expression
T	T	T
F	T	F
T	F	F
F	F	F

Example:

```
10 X=7:Y=10:Z=15
20 IF X=7 AND Y=10 THEN PRINT "TRUE"
30 IF X=7 AND Y=10 AND Z=15 THEN PRINT "TRUE"
40 IF X=5 AND Y=10 THEN PRINT "TRUE"
50 IF X=7 AND Y=12 THEN PRINT "TRUE"
60 IF X=5 AND Y=12 THEN PRINT "TRUE"
```

The statements in lines 20 and 30 will print the word "TRUE" when you run this short program. The statements in lines 40, 50 and 60 will not, because one or both of the expressions being tested are false. Both statements in line 20 are true, so the result is true and the statement to PRINT "TRUE" is executed. Note that in line 30, the AND statement is testing the truth of three statements (X=7 , Y=10, and Z=15). This works by evaluating the statements

two at a time.

For the OR expression, the result evaluates as true if either of the expressions are true, and if both expressions are false, then the result is false. The "truth table" for the OR statement looks like:

First Expression	Second Expression	Result Expression
T	T	T
F	T	T
T	F	T
F	F	F

Example:

```
10 X=7:Y=10:Z=15
20 IF X=7 OR Y=10 THEN PRINT "TRUE"
30 IF X=7 OR Y=10 OR Z=15 THEN PRINT "TRUE"
40 IF X=5 OR Y=10 THEN PRINT "TRUE"
50 IF X=7 OR Y=12 THEN PRINT "TRUE"
60 IF X=5 OR Y=12 THEN PRINT "TRUE"
```

The statements in lines 20, 30,40 and 50 will print the word "TRUE" when you run this short program because at least one of the statements is true. The statement in line 60 will not, because both expressions being tested are false. Note that in line 30, the OR statement is testing the truth of three statements (X=7, Y=10, and Z=15). This works by evaluating the statements two at a time.

Finally, if an expression evaluates to be true, then NOT <expression> is false. If the expression evaluates to be false, then NOT <expression> is true.

Example:

```
10 AB=10:BA=20
```

```
20 IF NOT(AB=BA) THEN PRINT "NOT EQUAL!"
```

Since AB is not equal to BA, the expression (AB=BA) is false. Thus, NOT(AB=BA) is true, and the program will print "NOT EQUAL".

HIERARCHY OF ARITHMETIC OPERATORS

Arithmetic expressions which include more than one operator are evaluated in a strict order. Certain operations are performed before other operations. This normal order can be modified by enclosing a portion of the expression consisting of two or more operands in parentheses. The portions of the expression enclosed in parentheses are always evaluated first, before working on parts of the expression outside the parentheses. Multiple levels of parentheses may be used. This is called nesting, and by using it just about any order of operator evaluation you wish can be achieved. Up to ten levels of nesting may be used.

Example:

```
X+(Y+2*(4/5)-4)
(((6^3)+4)/4)
```

When parentheses are not used, or within a given level of parentheses, the order of arithmetic operators is:

^	Exponentiation
-	Negation
* /	Multiplication and Division
+ -	Addition and Subtraction
< = >	Relational Operators
NOT	Logical NOT
AND	Logical AND
OR	Logical OR

As you can see, geoBASIC normally performs arithmetic operations first, then relational, then logical. If operators have the

same level or precedence (like * multiplication and / division) then the operators are evaluated from left to right. The normal order of precedence is maintained within parentheses, although any parts of the expression in the parentheses are evaluated before parts of the expression outside the parentheses.

STRING EXPRESSIONS

A string expression uses string operators and strings to produce a result which is another string. The first string operator is the concatenation operator, "+". This operator will combine the contents of two strings into one:

A$="HELLO" + " WORLD" (A$="HELLO WORLD")
H$=A$ + B$

Relational operators can also be used to compare strings. For the purposes of comparison, the letters of the alphabet are arranged in order such that A is greater than B which is greater than C, etc. Strings are compared by evaluating the characters in the string from left to right. If the first character in two strings are equal, then the next two characters are checked, and the next two, until either one string ends or a non-identical character is found. If all the characters in the strings are identical but one string is shorter than the other, the shorter string is considered to be "less than" the longer string. If the string comparison is true, then the result is -1 (or True, for the purposes of an IF statement), while if the comparison is false, then the result is 0 (or false, for the purpose of an IF statement).

Example:

"A" > "B" (true, result is -1)
"XY" = "YX" (false, result is 0)
A$<=B$

Note that this is the opposite of "normal" string evaluation in CBM BASIC.

Chapter 5—The geoBASIC Command Reference

ELEMENTS

The elements of the GEOS environment are very important in making geoBASIC both powerful and easy to use. The following is a brief description of these elements:

Menus

Menus refers to the line of words located at the top of the geoBASIC screen. Each of these words is called a submenu, and under each submenu is one or more items. Selecting one of these items will generally produce some result, such as saving your program, calling up the sprite editor, etc.

To select an item in a submenu, move the mouse pointer up to the submenu you want and press the left button. The submenu will "open up", showing the items under the submenu. To select an item, move the mouse pointer to it and press the left button. The submenu will then close up, removing the items from the screen, and act on your command.

Menus are available at the top of the text editing screen in geoBASIC to help you enter and work with your program. However, you can also design your own menus for use in your programs by using the menu editor, accessed by clicking on menu under the Utilities submenu. For more information on menus, see the chapter on Utilities.

Bitmaps

Bitmaps are pictures. They have a limited size, but are useful for showing pictorial representations of various items. For example, you could construct a bitmap showing a disk drive or printer. Bitmaps are useful with Dialog boxes and Icon Lists (see below). You may design your own bitmaps by selecting bitmap under the Utilities submenu. For more information about bitmaps,

see the chapter on Utilities.

Dialog Boxes

Dialog boxes are boxes which appear on the screen to give information to the user or to get information from the user. A dialog box can contain text, formulas, buttons and bitmaps. Generally, the user would click on a bitmap or button to select what he or she wants to do, with the text providing some instructions.

geoBASIC uses dialog boxes of its own, but you can also design your own dialog boxes for use in your programs by using the dialog box editor, accessed by clicking dialog under the Utilities submenu. For more information on dialog boxes, see the chapter on Utilities.

Sprites

Sprites are special graphic shapes which can move over the screen without disturbing the background picture, if any. Sprites can only be of a limited size, but can have up to three colors, can move across the screen, can be animated, and can have a velocity, initial position, and path set for them. You may design up to six sprites of your own, and can link multiple sprites so that the actions of several sprites are controlled by the motion of a single sprite.

To design your own sprites using the sprite editor, select sprite under the Utilities submenu. For more information on sprites, see the chapter on Utilities.

GEOBASIC COMMANDS

geoBASIC contains a wealth of commands, called keywords. Keywords are reserved, and appear in the section below in capital letters, listed in alphabetical order. You may not use keywords as variable names or imbed them in variable names. Study the explanations and example code carefully, and use this section as a reference in the future.

Terminology:

ARGUMENTS, also called parameters, can be associated with many keywords. The arguments appear in lower case with each keyword. Arguments can include filenames, variables, line numbers, expressions and math operators.

SQUARE BRACKETS [] show arguments which are optional. You may select any (or none) of the arguments shown.

ANGLE BRACKETS <> indicate that you MUST choose one of the arguments shown.

A VERTICAL BAR | separates items in a list of arguments. If the list appears in square brackets, then the choices are limited to those items listed, but the user still has the option of choosing any or none of the arguments. If the list appears in angle brackets, you MUST choose one of the items in the list.

ELLIPSIS ... A sequence of three dots means that an argument can be repeated more than once.

QUOTATION MARKS "" surround character strings, filenames and other types of expressions and arguments. When an argument is enclosed in quotation marks, the quotation marks must be included in the command.

PARENTHESES () When arguments are enclosed in parentheses, the parentheses must be included in the command.

VARIABLE refers to any valid BASIC variable name (Y, Z$, Q%, etc.)

EXPR refers to any valid numeric BASIC expressions, such as R*(4/T), etc.

STRING refers to a string constant, variable or expression.

ABS

FORMAT: ABS(<expr>)

This function returns the absolute value of the expression enclosed in the parentheses. The absolute value of an expression is equal to the expression itself if the expression evaluates to a number which is greater than zero. If the expression evaluates to a number which is less than zero, then the absolute value of the expression is the number without the negative sign, i.e., the absolute value of an expression is always positive.

Example:

```
10 X=-2:Y=3
20 PRINT X:PRINT ABS(X):PRINT Y:PRINT ABS(Y)
```

This sample program would print the values -2, 2, 3, 3. As you can see, the absolute value of the negative number X is positive.

AND

FORMAT: <expr> AND <expr>

The AND operator can be used for two purposes. As a mathematical (boolean) operator, it is used to combine two numbers together to produce a result. Each bit in the first expression is ANDed against the corresponding bit in the second expression. The bit in the result is equal to 1 if the bit in each expression is 1. If the bit in either expression is 0, then the bit in the result is zero. Thus, you get:

0 AND 0 = 0 1 AND 0 = 0 0 AND 1 = 0 1 AND 1 = 1

Each of the expressions must evaluate to a number between –32768 and +32767. If either of the expressions evaluate to a number outside this range, it will cause an ?ILLEGAL

QUANTITY error.

Example:

```
10 X=32007 AND 28761: PRINT X
```

This would produce the result 28673. To see why, convert each of the numbers to binary:

32007 is 0111110100000111 and 28761 is 0111000001011001.

ANDing each bit of the two numbers:

```
0111110100000111 AND
0111000001011001
---------------------------------------------------
0111000000000001    (binary) or
28673               (decimal).
```

The other way to use the AND operator is to test the truth of two expressions. Each expression is generally an IF statement (see IF). The result evaluates as true only if both of the expressions are true, and if either expression is false, then the result is false. The "truth table" for the AND statement looks like:

First Expression	Second Expression	Result Expression
T	T	T
F	T	F
T	F	F
F	F	F

Example:

```
10 X=7:Y=10:Z=15
20 IF X=7 AND Y=10 THEN PRINT "TRUE"
30 IF X=7 AND Y=10 AND Z=15 THEN PRINT "TRUE"
40 IF X=5 AND Y=10 THEN PRINT "TRUE"
```

```
50 IF X=7 AND Y=12 THEN PRINT "TRUE"
60 IF X=5 AND Y=12 THEN PRINT "TRUE"
```

The statements in lines 20 and 30 will print the word "TRUE" when you run this short program. The statements in lines 40, 50 and 60 will not, because one or both of the expressions being tested are false. Both statements in line 20 are true, so the result is true and the statement to PRINT "TRUE" is executed. Note that in line 30, the AND statement is testing the truth of three statements (X=7 , Y=10, and Z=15). This works by evaluating the statements two at a time. First, the truth of X=7 AND Y=10 is tested. In this example, the result is true. Then, the result of this test (TRUE) is tested with the third statement, Z=15. Since Z is 15, this evaluates to: TRUE AND TRUE, which is TRUE. Larger groups of statements can be tested for truth in this way, and statements can be grouped together using parentheses:

```
10 IF (X=5 AND Y=10) AND (Z=20 AND Z*Y=200) THEN....
```

In this example, the truth of X=5 AND Y=10 is evaluated and stored. Then the truth of Z=20 AND Z*Y=200 is tested and stored. Finally, the two stored results are tested against each other for the final result.

When a statement evaluates as FALSE, the value 0 is assigned to the result, while if the statement evaluates as true, the value of –1 is assigned to the result. Your program can determine the numerical value that the expression evaluates to by equating the expression to a variable:

Example:

```
10 X=10:Y=20
20 RES1=(X=20)
30 RES2=(X=10) AND (Y=20)
```

The variable RES1 will be zero, since the statement (X=20) is false. The variable RES2 will be –1, since both statements (X=10) and (Y=20) are true. As above, you can combine more than two expressions and evaluate the numerical result.

APPEND

FORMAT: APPEND <recordnum>

This command adds a new record to a VLIR file. Recordnum is either a number or a numeric variable that points to the record that will be appended to. For example:

before: **0 1 2 3 4 5 ...**

after an APPEND 2: **0 1 2 3 4 5 6 ...**
^
— new record

All records after the appended record are moved up one record. If the last record would exceed 127 then an OUT_OF_RECORDS error will occur. There is a bug in this command that prevents APPENDing to record 126. APPEND will do an implicit PTREC to the new record (see also INSERT).

ASC

FORMAT: ASC(<string>)

The ASC function will return the ASCII code of the first character of the string. The expression <string> may be a string constant or string variable. If there are no characters in the string, then an ?ILLEGAL QUANTITY error will result. If the expression in the parentheses is not a string (a number or letter) then a ?TYPE MISMATCH ERROR results. The number returned for the ASCII value will be between 0 and 255.

Example:

```
10 PRINT ASC("A")
20 PRINT ASC("HELLO")
30 J$="HELLO"
40 PRINT ASC(J$)
```

This short program will print 65 for line 10 (ASCII value of

"A"), 72 for line 20 (ASCII value of "H", the first letter in "HELLO") and also a 72 for the result of line 40.

ATN

FORMAT: ATN(<expr>)

This function returns the arctangent of the expression, which must evaluate to a number between -PI/2 and PI/2. The arctangent is measured in radians.

Example:

```
10 PRINT ATN(2)
```

returns 1.10714872

BITMAP

FORMAT: BITMAP(<string>),(<expr>),(<expr>)

This command puts a bitmap on the screen. The bitmap must have been created using the bitmap editor from the Utility submenu or pasted in from a photoscrap beforehand. The <string> is the filename that the bitmap was stored with. The first <expr> specifies the X coordinate of the upper left corner of the bitmap and must evaluate to a number between 0 and 39. The second <expr> is the Y coordinate of the upper left corner of the bitmap and must evaluate to a number between 0 and 199. If you place a bitmap so that the right edge of the bitmap extends past the right edge of the screen, the extra portion will be clipped (not be visible). If you place a bit map so that the bottom extends past the bottom of the screen, that portion of the bitmap won't be visible on the screen.

Example:

```
10 BITMAP "pict",10,20
```

BUTTON

FORMAT: BUTTON <expr>

This command executes the subroutine which is given by the expression every time the user clicks the left mouse button even if the button is over a menu or icon. The expression defines the line number to GOSUB to. The subroutine must end with a RETURN.

Example:

```
10 BUTTON 1000 . . 1000 PRINT "HI":RETURN
```

CALL

FORMAT: CALL <expr> [,<expr>, <expr>, <expr>, <expr>]

This command calls a machine language subroutine. The first expression is the only one which is required. It gives the address of the machine language routine. The rest of the expressions are optional, and must evaluate to integers between 0 and 255. They are used for (in order) passing values to the accumulator, X register, Y register and the status flags. Using CALL makes it possible to call GEOS routines which are not directly supported. The memory addresses and details of GEOS routines are listed in the GEOS Programmer's Reference Guide. As an example, the GetScanLine routine (pg. 102 of the Programmer's Reference Guide) can be called, where the variable "X" contains the scan line number:

Example:

```
10 CALL 49468,0,X
```

CHR$

FORMAT: CHR$(<expr>)

This function converts an ASCII code (such as one returned by the ASC function, above) to its character equivalent. The expression must evaluate to an integer between 0 and 255. If it does not, an ?ILLEGAL QUANTITY error will result. When used with the PRINT command (see below), CHR$ allows you to PRINT using different styles. These styles are:

CHR$(14)	**Turns on underlining**
CHR$(15)	**Turns off underlining**
CHR$(18)	**Turns on reverse video characters**
CHR$(19)	**Turns off reverse video characters**
CHR$(24)	**Turns on bold printing**
CHR$(25)	**Turns on italics printing**
CHR$(26)	**Turns off all effects (returns to normal printing)**

Example:

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

returns the character "A".

NOTE: Do not print the values 1 thru 7 or 29 thru 31 with the CHR$ function. Doing so will cause a SYSTEM ERROR.

CLOSE

FORMAT: CLOSE

This statement closes the data file which was opened using the OPEN command (see below). Only a single file can be OPEN at once, so you must use CLOSE to close any open file before OPENing another file.

Example:

```
10 CLOSE
```

CLS

FORMAT: CLS

This command clears the current window on the graphic screen, where all output takes place during a program. If no current window has been specified, then the entire graphic screen will be cleared. It is useful to begin any program with CLS to give yourself a clear screen to work on.

Example:

```
10 CLS
```

COLRECT

FORMAT: COLRECT <expr>,<expr>,<expr>,<expr>

This command draws a colored rectangle on the screen. The actual color is set by the command SETCOL (see below). The four expressions are the X and Y coordinates of the upper left corner of the rectangle and the X and Y coordinates of the lower right corner of the rectangle. The X coordinate expressions must evaluate to a number between 0 and 39, while the Y coordinate expressions must evaluate to a number between 0 and 24. Menus and Dialog boxes will appear in the colors set with SETCOL. geoBASIC does not automatically change the colors beneath menus and dialog boxes.

Example:

```
10 COLRECT 10,10,20,20
```

COS

FORMAT: COS(<expr>)

This function calculates the cosine of the expression, which

must evaluate to a number. The number is the angle in radians.

Example:

```
10 PRINT COS(20)
20 Y=COS(Z*PI/180): REM CONVERT DEGREES TO RADIANS
```

CREATE

FORMAT: CREATE <filename$>[,<drivenum>]

This command creates a VLIR file on the disk. A VLIR file is composed of a collection of records, each of which may have a maximum size of 32K bytes. There may be up to 128 of these records in the file (numbered 0 to 127). The filename parameter may be a string variable or quoted string of any length but only the first 16 characters are significant. Drivenum is an optional parameter and specifies the device number to create the file on. This number can range from 8 to 11. If this parameter is omitted the current drive will be used.

Unless changed with the HEADER command, CREATE will create only files of type 'BASIC DATA', with an empty permanent name string. DO NOT USE CREATE IF THE THIRD PARAMETER OF THE HEADER COMMAND IS ZERO! CREATE will not create sequential files properly!

Note that CREATE will leave the file in an open state so there is no need to issue an OPEN command prior to accessing the file.

Example:

```
10 CREATE "TEST" : REM CREATES 'TEST' ON THE CURRENT
   DRIVE
10 CREATE A$,9 : REM CREATES FILE NAMED IN A$ ON DEVICE #9
```

DATA

FORMAT: DATA <list of constants>

DATA statements are followed by a list of data items separated by commas whose values are read into variables by the READ statement (see READ, below). The data items can be numeric or strings and strings do not have to be enclosed in quotes unless the string contains a space, comma, colon, shifted letters, graphics or cursor control characters. Two commas with nothing between them will be entered as a zero if received by a numeric variable or as an empty string if received by a string variable.

All the data statements in a program are treated as one continuous list, regardless of positioning in the program. The data in the statements are read in sequence from left to right, starting with the lowest numbered line and proceeding to the highest. This order of reading can be modified using the RESTORE statement (see below). The data in the DATA statements must match the type of the variable it is being read in to. Numeric data is read into numeric variables, and strings are read into string variables. If the READ statement encounters data which doesn't match the type of the variable, an error will result. Strings do not have to be enclosed in quotes unless they contain special characters. To include spaces, commas, colons and semicolons in string data, it must be enclosed in quotes.

Example:

```
10 DATA 100,200,ABCDEFG
20 DATA 224.5,DAVID,"HELLO WORLD","YES, SIR"
```

DBFILE

FORMAT: DBFILE <string>

This command places a special dialog box on the screen for the user to choose the name of a file on disk for use. The dialog box

has a scrolling list of files on the left side, just like the dialog box which appears when you first start geoBASIC. If there are more files than can be shown in the file box, a pair of arrows will appear near the bottom of the box. To scroll through the list of available files, move the mouse pointer to the up or down arrows and click the left button. The user can click on a filename to select it. This removes the dialog box from the screen, restoring whatever was hidden by it. The name of the file chosen by the user is returned in the <string>.

Unless changed by the HEADER command, only files with type BASIC DATA and with permanent file name of "" (null string) will be shown in the file box.

Once the DBFILE command has been successfully executed, the file is opened, and so can be read from (see DREAD) or written to (see WRITE).

Example:

```
10 DBFILE A$   :REM RETURNS THE NAME OF THE OPENED FILE
   IN A$
```

DBSTRN

FORMAT: DBSTRN <string>,<string>

This function places a special one-line dialog box on the screen to get input from the user. The first <string> is a prompt which is printed in the dialog box. The second <string> must be a variable — the user's typed input is returned in this variable when the user presses [RETURN] after typing in the requested information. The dialog box is then automatically removed from the screen and whatever was obscured behind it is restored.

Example:

```
10 DBSTRN "Your name",A$
```

DEF FN

FORMAT: DEF FN<name> (<variable>) = <expr>

This statement sets up a user-defined function that can be used later in the program. The function can consist of any mathematical statement or expression, and must be limited to one line. The <name> of the function must follow FN and can be any alphanumeric variable name beginning with a letter. This name is used later when the function is referenced. The <variable> must be included for proper syntax, but does not need to be used in the function definition. When the function is called (see FN, below) with a variable or constant in parentheses, the value of the variable or constant it is called with replaces the variable in the function definition everywhere it appears. The DEF FN statement must be executed during the course of running the program before it becomes active.

Example:

```
10 Q=5:R=4
20 DEF FN ABC(X)=X*3
30 DEF FN QQQ(Y)=Q+R/4
40 BB=FNABC(10):PRINT BB
50 CC=FNQQQ(R*R):PRINT CC
```

There are several things to notice in this example. First, the DEF FN statements must be executed before the FN statements which call them. Also, the function (as called using the FN statement) is treated just like any other math function such as COS (see above). When the function is called, its value is automatically calculated. The statement on line 40 will print 30 for the value of BB, since 10 is substituted for the variable X in the function definition on line 20. If a different variable or constant was used on line 40, then you would get a different result. The statement on line 50 will print 5 for the value of CC. This result will be the same no matter what variable is included in the function call, since the formula in line 30 doesn't use the variable.

DELETE

FORMAT: DELETE <expr>

This command deletes the record number given by the expression from a file created using the CREATE command (see above) or opened using DBFILE or OPEN. All records in the file with higher numbers than the deleted record will be moved down one.

Example:

```
10 DELETE 3
```

DELPROC

FORMAT: DELPROC <expr>

The PROCESS command (see below) can set up a process subroutine which will execute periodically. The DELPROC command stops a process which was started by the PROCESS command from running. The expression is the line number of the process subroutine which you no longer want to run. Since a maximum of eight processes can be running at any time, this command allows you to turn off processes so that others can be started, if you wish.

Example:

```
10 PROCESS @FLASH, 10
.
.
100 DELPROC @FLASH
```

DIALOG

FORMAT: DIALOG <string>, [<variable>]

Places a dialog box designed using the Dialog Box Editor on the screen. The <string> can be a string constant or string variable, and specifies the name of the dialog box to place on the screen. This is the name used when you constructed the dialog box using the Dialog Box Editor. If the second parameter is used, it must be a numeric variable. The number corresponding to the icon or bitmap that the user clicked on to exit the dialog box is returned in the variable. These numbers and their corresponding icons are explained in the Dialog Box Editor section, above. When the user clicks on an icon to remove the dialog box from the screen, whatever was obscured by the dialog box is restored to the screen.

Example:

```
10 DIALOG "TEST",A
20 DIALOG A$,B
```

DIM

FORMAT: DIM <variable>(<subscripts>)[,<variable>(<subscripts>)...]

Before arrays of variables can be used, the dimensions of the array must be established using the DIM statement. The <variable> name can be any legal variable name. The name must follow the rules for variable names: the array is automatically an array of floating point numbers unless the "$" character is used at the end of the array name to indicate a string array or the "%" symbol is used at the end of the variable name to indicate an array of integers.

The <subscripts> argument establishes the limits of the array and how many dimensions it will have. One subscript is used for each dimension, and the subscripts specifying the limits in each dimension must be separated by commas. Up to 255 dimensions may be used, subject only to the requirement that there be enough memory to hold the array. An array with more than one subscript is known as a matrix. Each subscript establishes the number of

elements in the array for that dimension. The lowest element number is 0, and the highest allowed is 32767. Arrays are numbered from 0 to N, where N is the maximum value specified by the subscript in the DIM statement. Since the lowest numbered subscript is 0, there is actually one more element in the array than the value of the subscript. Further, each element of a string array (variable name ends in "$") can hold a string.

The DIM statement for an array must be executed once and only once during the course of program execution. Any attempt to reexecute a DIM statement will result in a REDIMed ARRAY ERROR. If an array is used in the program which was never dimensioned, it automatically is dimensioned to 11 elements in each dimension used in the first reference.

Example:

```
10 DIM A(100):REM 101 elements (0-100)
20 DIM B(4,5),Q(3,4,5)
30 DIM B$(100)
```

You need to be careful not to run out of memory while DIMensioning arrays. Arrays have the following memory requirements:

5 bytes for the array name
2 bytes for each dimension
2 bytes/element for integer variables
5 bytes/element for normal numeric variables
3 bytes/element for string variables
1 byte for each character in each string element.

DPEEK

FORMAT: DPEEK <expr>

This function returns the word (two-byte) value at the location given by <expr>. <Expr> must evaluate to an integer between 0 and 65535. The contents will always be a number between 0 and 65535.

Example:

```
10 A=DPEEK(5055)
```

DPOKE

FORMAT: DPOKE <location>,<expr>

Places the word (two-byte) number given by <expr> into the memory location given by <location>. The low byte of <expr> is placed in <location>, while the high byte is placed at <location> +–1. Both <location> and <expr> must be between 0 and 65535.

The DPOKE command writes the <expr> directly into a memory location. Extreme care should be exercised when using this command, since putting the wrong value into a memory location could cause your computer to lock-up.

Example:

```
10 A=2040:B=54320:DPOKE A,B
```

DREAD

FORMAT: DREAD <variable$>[,<variable$>,...]

This command is used to fill variables with information from a disk file. The file being read from must have been previously opened with the OPEN, CREATE, or DBFILE commands. While numeric variables are permitted they are not recommended because if the information coming in from the disk is not numeric an error will result. Use string variables and cast them to numbers with the VAL() command instead. Each string in the file must be terminated with a carriage return or a comma. An error will occur if the string is longer than 255 characters or if you try to read from an empty record (see also RDBYTE).

END

FORMAT: END

This statement stops program execution, returns to the text editor and waits for a keypress. Once a key has been pressed, it displays the READY message on the screen. It is not necessary to use any END statements, although it is good programming practice to conclude the program with one. There can be any number of END statements throughout a program to halt execution.

Example:

```
10 PRINT "DO YOU WANT TO QUIT?"
20 INPUT ANS$
30 IF ANS$="YES" THEN END
40 REM REST OF PROGRAM
```

EOF(0)

This function is used to signal the end of a disk record. The argument may be any variable or a number. If the last DREAD or RDBYTE returned the last character of the record, or if PTREC is used on an empty record then a TRUE (-1) is returned. Otherwise EOF(0) returns zero.

Remember that there is a difference between an empty record and an unused record. An empty record is one that was created with INSERT or APPEND but nothing was written to it. An unused record has never been accessed at all. PTREC will generate an error if you try to point to an unused record, preventing the use of EOF()!

Example:

```
10 OPEN "MYFILE"
20 PTREC 0
30 WHILE NOT EOF(0)
40 RDBYTE A$
50 A = ASC(A$)+CHR$(0)
60 IF A > 31 OR A = 13 THEN PRINT A$;
70 LOOP
80 CLOSE
90 END
```

EXP

FORMAT EXP(<expr>)

This function calculates the base of the natural logarithms (e, equal to approximately 2.71828) raised to the power given by the argument <expr>. A value for <number> greater than 88.0296919 will cause an ?OVERFLOW ERROR.

Example:

```
10 X=Y*EXP(10*X+.5)
```

FN

FORMAT: FN<name>(<expr>)

This function returns the value of the formula previously defined using DEF FN (see above).The FN function must be executed after the DEF FN call which defines it. The <expr> can be a variable, constant or expression whose value is substituted into the place of the variable in the DEF FN formula when the formula is calculated. The FN function works just like any ordinary function and its value is calculated automatically when it is called, using the value of the <expr> specified.

Example:

```
10 Q=5:R=4
20 DEF FN ABC(X)=X*3
30 DEF FN QQQ(Y)=Q+R/4
40 BB=FNABC(10):PRINT BB
50 CC=FNQQQ(R*R):PRINT CC
```

The statement on line 40 will print 30 for the value of BB, since 10 is substituted for the variable X in the function definition on line 20. If a different variable or constant was used on line 40, then you would get a different result. The statement on line 50 will print 5 for the value of CC. This result will be the same no matter what variable is included in the function call, since the formula in line 30 doesn't use the variable.

FONT

FORMAT: FONT<string>,<expr>

This command specifies what font to use. If the font is not in memory, then the disk will be searched for the font, and it will be automatically loaded if it is found. If it is not found, then the default system font will be used instead. The <string> specifies the name of the font to use, while the <expr> specifies the size of the font. The size is in points, with 72 points to the inch. Fonts included with GEOS are:

Font	Sizes
BSW	9
University	6,10,12,14,18,24
California	10,12,13,14,18
Roma	9,12,18,24
Dwinelle	18
Cory	12,13

20 additional fonts are available on the GEOS Font Pack 1. If a font is too large, there may not be enough memory to contain it.

You would then get an ?OUT OF MEMORY error.

When printing fonts on the screen, the carriage returns (used to get to the next line) are the same size as the font. Carriage returns are produced with a PRINT statement which is NOT followed by a semi-colon. A large-sized font will produce a large carriage return (large distance between the current line and the next line). Changing font sizes just after doing a carriage return can produce some strange results. Switching from a small font to a large one will cause the large font to overprint the line on which the small font is located. Switching from large font to a small one can cause wide gaps between lines of text.

Example:

10 FONT "University",10

FOR...TO...[STEP]...NEXT

FORMAT: FOR <variable>=<start> TO <end> [STEP <increment>]

.

.

NEXT [<variable>, <variable>...]

This series of commands establishes a loop that repeats the statements contained in the loop for a set number of times. All statements between the FOR and the NEXT statements are repeated. The <variable> is used as a counter for the loop. It must be a floating point variable (may not be an integer or string variable). It starts out with the value specified by <start>. All the statements up to the NEXT statement are then executed. When the <NEXT> statement is encountered, the value of the loop variable is changed by the amount specified by <increment> in the STEP statement. If the STEP statement is not used, then the <increment> is automatically set to 1. The value of the variable is compared to <end>. If the loop variable has exceeded the value of <end> then execution of the program continues with the next statement past

the NEXT statement. If the loop variable has not exceeded the value of <end>, then the program loops back and resumes execution with the statement following the FOR statement. If the <increment> value in the STEP statement is negative, then execution continues within the loop only as long as the loop variable is greater than the value of <end>.

The NEXT statement indicates the end of the loop. If the optional <variable> is used with the NEXT statement, it must be the same variable name as the loop variable established in the FOR statement. One FOR...NEXT loop may be contained within another. This is known as nesting. You may nest loops up to nine deep. A single NEXT statement can terminate several nested loops. If the <variable> is not used with the nested NEXT statement, then the NEXT statement will terminate the last started loop. If the <variable> is used with the nested NEXT statement, you must either use separate NEXT statements to terminate each nested loop or else use multiple <variable>s with a single NEXT statement, being careful that the variables are in the proper order: the last loop to start (the inside loop) must be the first loop to end. Loops may not cross one another. See the examples for illustrations.

Example:

```
10 FOR X=1 TO 10:REM NO STEP STATEMENT
20    PRINT X:NEXT

10 FOR X=1 TO 20 STEP .5
20    FOR Y=10 TO 0 STEP -1
30       PRINT X*Y+5
40    NEXT Y:REM FINISH THE INSIDE LOOP
50 NEXT X:REM FINISH THE OUTSIDE LOOP

10 FOR X=1 TO 20 STEP .5
20    FOR Y=10 TO 0 STEP -1
30       PRINT X*Y+5
40 NEXT Y,X:REM ONE NEXT WITH TWO VARIABLES
```

```
10 FOR X=1 TO 20 STEP .5
20    FOR Y=10 TO 0 STEP -1
30        PRINT X*Y+5
40    NEXT:REM NO VARIABLE—TERMINATES INSIDE LOOP
50 NEXT:REM NO VARIABLE—TERMINATES OUTSIDE LOOP
```

FRE

FORMAT: FRE(<expr>)

This function tells you how much memory is left in the computer for you to use for your program and variables. If the program tries to use more memory than is available, an OUT OF MEMORY error will result. The <expr> can be anything since it is not used except for syntax purposes.

Example:

```
10 PRINT FRE(0):REM FREE MEMORY
```

FRECT

FORMAT: FRECT <expr>,<expr>,<expr>,<expr>

This command draws a framed rectangle on the screen. A framed rectangle is a rectangle which is not filled in, that is, it consists only of the four lines which form the frame. The four expressions are the X and Y coordinates of the upper left corner of the rectangle and the X and Y coordinates of the lower right corner of the rectangle. The X coordinate expressions must evaluate to a number between 0 and 319, while the Y coordinate expressions must evaluate to a number between 0 and 199.

The value of SETCOL sets the line pattern of the four lines which make up the frame of the rectangle. The 8 bits which make up the number passed to SETCOL can be either On or Off. If the bit is On, then it shows up in the line, and if the bit is Off, then it is

off in the line. For example, the number 255 (binary 11111111) has all bits on, so the line drawn with FRECT will be solid. 85 (binary 10101010) will produce a dashed line, with every other pixel of the line being On. Using a value of zero will cause a white line to be drawn on any present dark background.

Example:

```
10 SETCOL 255
20 FRECT 10,10,20,20
```

GET

FORMAT: GET<variable list>

This statement reads any key you press. The variables in the <variable list> will receive the values of the keys pressed. String or numeric variables may be specified in the <variable list>, but if a numeric variable is specified and you press a letter key, then an error will result. It is better to use only string variables and convert strings to numbers (see VAL, below) where necessary. If no key is pressed, then the variable will be empty and the program continues without waiting. The GET statement may be put into a loop so that you can check for an empty result.

Example:

```
10 GET A$:IF A$="" THEN GOTO 10:REM WAIT FOR NONEMPTY A$
20 PRINT A$
```

GOSUB/RETURN

FORMAT: GOSUB <expr>

The GOSUB statement transfers control of the program to the line specified by <expr>. The <expr> parameter may be a numeric constant, variable, expression or label. The block of statements beginning at <expr> is then executed. When the RETURN state-

ment is encountered, program control returns to the statement following the GOSUB. The computer remembers where the new program segment was called from and returns there when the block of statements located between the chosen line and the RETURN statement, known as a subroutine, is finished. Subroutines are of primary use for blocks of the program which are repeated many times. Instead of putting the code in the program many times, the program can simply GOSUB to the line where the subroutine begins each time that block of code needs to be executed.

The contents of the subroutine block can use any valid geoBASIC statements, including calls to other subroutines. However, since the address that the subroutine must RETURN to is stored in a limited section of memory, there is a definite limit to how deep you can nest subroutine calls. If you try to nest too many GOSUBs, you will get an OUT OF MEMORY error — even though there may be plenty of memory left for the rest of your program.

Example:

```
10 FOR N=1 TO 10
20     GOSUB @DOIT:REM THE SUBROUTINE CALL
30 NEXT N
40 END
100 @DOIT:PRINT EXECUTION # ";N:REM THE SUBROUTINE
110 RETURN:REM END OF THE SUBROUTINE
```

Line 40 is very important. If it weren't there, then after the subroutine was executed by the GOSUB in line 20 and the loop finished in line 30, the program would continue to run at line 100. However, when the RETURN in line 110 was encountered, the error message RETURN WITHOUT GOSUB ERROR in 110 would result. That is because the computer would not know where to RETURN to since there had been no corresponding GOSUB call.

GOTO

FORMAT: GOTO <expr>

This statement allows the program to jump to the line number specified by <expr> and continue execution there. The <expr> parameter may be a numeric constant, variable, expression or label.

Example:

```
10 PRINT "GOTO STATEMENT DEMONSTRATION"
20 GOTO @DOIT:REM COULD ALSO BE GOTO 100 OR GOTO 10*10
30 GOTO 30:REM PRESS RUN/STOP TO STOP PROGRAM
40 REM PROGRAM WILL NEVER GET HERE
100 @DOIT: PRINT "THIS IS LINE 100"
110 END
```

HEADER

FORMAT: HEADER <expr>,<string>[,<expr>]

The HEADER command allows you to specify the type of file and permanent file name of the files which will appear when using DBFILE or that will be created using CREATE. If the HEADER command is not used, then only files matching type BASIC DATA and with a permanent file name of "" (null string) will appear when using DBFILE or be created when using CREATE. The first <expr> must evaluate to a valid file type. Valid file types are:

BASIC (1), ASSEMBLY (2), DATA (3), SYSTEM (4), DESK_ACC (5), APPLICATION (6), APPL_DATA (7), FONT (8), PRINTER (9), INPUT_DEVICE (10), DISK_DEVICE (11), SYSTEM_BOOT (12), TEMPORARY (13) and AUTO_EXEC (14).

Most of these file types would never need to be used or accessed by a user. For more information on the different file

types, see page 398 of the GEOS Programmers Reference Guide.

The <string> is the permanent file name to match. If used, then only files which match this permanent file name will appear in the dialog box. The permanent file name is used by GEOS to help identify what application a file belongs to. For example, it is by the permanent file name that GEOS can tell the difference between GeoPaint and GeoWrite files. The permanent file name of any file can be seen from the desktop. Click on the file you are interested in to highlight it, then drop down the File submenu and click on Info. The name which appears in the Info box next to Class is the permanent file name.

The optional last <expr> is the file structure. It must evaluate to either 0 (sequential) or 1 (VLIR).

Example:

```
10 HEADER 3,"",1
```

ICON

FORMAT: ICON <string>

Places a group of icons on the screen. The group of icons must have been designed previously using the Icon List Editor. The <string> can be a string constant or string variable, and specifies the name of the Icon List to place on the screen. This is the name used when you constructed the Icon List using the Icon List Editor. See MAINLOOP for an example of how your program should respond when the user clicks on one of the icons you have placed on the screen. If multiple calls are made to the ICON command, only the last set of icons loaded can be clicked on, even if earlier-loaded icons are still visible on the screen.

Example:

```
10 ICON "MYCON"
```

IF...THEN...

FORMAT: IF <expr> THEN <statements>

This statement gives geoBASIC the ability to make decisions based on the outcome of the expression. <Expr> can be any mathematical formula including variables, strings, numbers, comparisons and logical operators. If the expression is true, then the statements following THEN are executed. If the expression is

Example:

```
10 REM DEMO NUMBER CHOOSING GAME
20 PRINT "I WILL PICK A NUMBER BETWEEN 1 AND 10":CNT=0
30 A=INT(RND(10))+1
40 PRINT "WHAT IS YOUR GUESS?":INPUT Q:CNT=CNT+1
50 IF Q<1 OR Q>10 THEN PRINT "NO. OUT OF RANGE":GOTO 40
60 IF Q=A THEN GOTO 90
70 IF Q<A THEN PRINT "GUESS IS TOO LOW":GOTO 40
80 PRINT "GUESS IS TOO HIGH":GOTO 40
90 PRINT "YOU GOT IT IN ";CNT; " TRIES"
100 PRINT "WANT TO PLAY AGAIN (Y/N):INPUT A$ 110 IF A$="Y"
    THEN GOTO 20
120 PRINT "GOODBYE...":END
```

Line 20 prints out a message and zeroes out the counter. Line 30 uses the RND function to choose a number between 1 and 10 (see RND, below). Line 40 gets your guess. Line 50 uses a compound expression to make sure that your guess is in the proper range, and prints a message and GOTOs line 40 if it is not. Line 60 jumps to line 90 if you got the right answer. Line 70 tests to see if your guess is less than the number, prints the message and jumps to the right line number if it is. Notice how line 80 works. If lines 60 and 70 didn't cause a jump to another line, then your guess MUST be higher than the right number, so line 80 prints this message and jumps back to line 40 to get your next guess. Line 90 prints out the winning message, and line 100 checks to see if you want to play again. If you type in "Y", then line 110 will cause a branch back up to line 20 to start over. If you don't type a "Y",

then the program falls through to line 120, which prints a message and ENDs the program.

INPUT

FORMAT: INPUT ["<prompt>";]<variable list>

This statement receives input from you, and places what you type into the variables in the variable list. When the INPUT statement is encountered in the program, the program stops and a question mark is placed on the screen. Type in your data and press [RETURN]. The INPUT command may be followed by any text enclosed in quotes. This text will be printed on the screen, followed by the question mark. The text is helpful in reminding you what sort of information the program needs from you. The semicolon following the prompt text MUST be used in the statement if the prompt text is used.

The <variable list> may contain one or more variable. If only a single variable is used, then you can just type in the value and press [RETURN]. If more than one variable is used, then type in the appropriate number of values, separated by commas. If you type in too few values, a "??" will appear on the next screen line to prompt you to type in additional values. If you type in too many values, the ?EXTRA IGNORED message will appear, meaning that the extra items you typed were not placed into any variables. Note that, since the values you type in are separated by commas, the values themselves may not contain any commas. Also, if the current variable in the variable list is a numeric variable and you type in a string, you will get the ?REDO FROM START message. You must then type in a number for that variable. If you just press [RETURN] at the INPUT prompt, the old value of the variable is maintained (the variable value doesn't change).

Example:

```
10 INPUT VLUE,START, FINISH
20 INPUT SVL$:REM GET A STRING
30 INPUT "WHAT NUMBER"; NMBR
```

INSERT

FORMAT: INSERT <recordnum>

This command adds a new record to a VLIR file. Recordnum is either a number or a numeric variable that points to where the record will be inserted. For example:

before: **0 1 2 3 4 5 ...**

after an INSERT 2: **0 1 2 3 4 5 6. . .**

^

- - - new record

All records after the inserted record are moved up one record. If the last record would exceed 127 then an OUT_OF_RECORDS error will occur. There is a bug in this command that prevents INSERTing to record 126. INSERT will do an implicit PTREC to the new record (see also APPEND).

INT

FORMAT: INT(<expr>)

This function returns the integer value of the expression, which must evaluate to a number. If the expression is a positive number, the fractional portion of the number is left off. If the expression is less than zero, any fractional part causes the next lower integer to be returned.

Example:

```
100 PRINT INT(10.5), INT(-10.5)
```

These statements return the values 10 and –11, respectively.

INVRECT

FORMAT: INVRECT <expr>,<expr>,<expr>,<expr>

The command inverts screen pixels in the rectangle defined by the four expressions. All pixels which are On are turned Off, and all pixels which are Off are turned On. The four expressions are the X and Y coordinates of the upper left corner of the rectangle to invert and the X and Y coordinates of the lower right corner of the rectangle. The X coordinate expressions must evaluate to a number between 0 and 319, while the Y coordinate expressions must evaluate to a number between 0 and 199.

Example:

```
10 INVRECT 10,10,20,20
```

LEFT$

FORMAT: LEFT$(<string>,<expr>)

This function returns the left-most <expr> characters of the string <string>. The <expr> parameter can be an expression which evaluates to an integer between 0 and 255. If the integer is greater than the length of the string, the entire string is returned. If the integer is 0, then a null (empty) string is returned.

Example:

```
10 A$="BERKELEY SOFTWORKS"
20 B$=LEFT$(A$,8):PRINT B$
30 C$=LEFT$("HELLO WORLD",5):PRINTC$
```

This example program would print "BERKELEY" and "HELLO".

LEN

FORMAT: LEN(<string>)

Returns the length of <string>. Blanks and unprintable characters are included in this count.

Example:

```
10 A$="BERKELEY SOFTWORKS"
20 B=LEN(A$):C=LEN("HELLO WORLD")
30 PRINT B:PRINT C
```

This example would print the values 17 and 11.

LINE

FORMAT: LINE <expr>,<expr> TO <expr>,<expr>

This command draws a line on the screen. The first two expressions (preceding TO) are the X and Y coordinates of the beginning of the line, and the last two expressions (after TO) are the X and Y coordinates of the end of the line. The X coordinate expressions must evaluate to a number between 0 and 319, and the Y coordinate expressions must evaluate to a number between 0 and 199. The color of the line is set by SETCOL(below).

Example:

```
10 LINE STX,STY,EX,EY
20 LINE 10,10,20,20
```

LIST

FORMAT: LIST[[<first line>],[<last line>]]

The LIST command allows you look at the geoBASIC program currently in memory. The screen editor can be used to edit portions of the program once these portions have been placed on the screen.

If the program is too long to fit on the screen, the lines at the top of the screen will scroll off the top as new lines are added at the bottom. To suspend this scrolling, hold down the F5 key. Releasing the F5 key will allow the LIST to resume again. To break in to a LIST statement, press [RUN/STOP].

If the LIST command is used without any parameters, the entire program in memory is listed. The <first line> and <last line> parameters may be constants or labels. If the first line number is given followed by a comma (,), then all lines from the specified line to the end of the program will be listed. If only the last line is given, preceded by a comma, then all lines from the beginning of the program to the specified line will be listed. If both the starting and ending line numbers are given, separated by a comma, then all lines between and including the specified lines will be listed. If just a single line number is specified after the LIST statement, then just that line will be LISTed.

Example:

LIST	**(LISTS WHOLE PROGRAM IN MEMORY)**
LIST 100	**(LISTS LINE 100 ONLY)**
LIST 100,	**(LISTS LINE 100 TO END OF PROGRAM)**
LIST ,100	**(LISTS FROM BEGINNING OF PROGRAM THROUGH LINE 100)**
LIST 100,500	**(LISTS LINES 100 THROUGH 500)**
LIST @START,@END	**(LISTS FROM LINES @START TO @END)**

LOAD

FORMAT: LOAD <filename>,<expr>,[<expr>]

This statement reads the contents of a file from disk into memory. The filename identifies the name of the file you want to load. The filename may be contained in a string variable. If the filename specified is not found, the ?FILE NOT FOUND error message will result.

The first expression is the memory address to load the file into. It must evaluate to a number between 0 and 65535. The second expression specifies the device number that the file is to be loaded from. It must evaluate to a number between 8 (first disk drive) and 11. If the expression is left out, the current drive will be used.

Example:

LOAD "PIC",40960 **(load "PIC" from current drive to screen memory)**
LOAD B$,32768,8 **(loads the filename given by B$ from disk. The file will be loaded to memory location 32768.)**

LOG

FORMAT: LOG(<expr>)

Returns the natural logarithm (log to the base of e) of the expression, which must evaluate to a number greater than zero. An ?ILLEGAL QUANTITY error will occur if the number is less than or equal to zero.

Example:

```
10 PRINT LOG(10/7)
```

MAINLOOP

FORMAT: MAINLOOP

This simple command is the heart or "main loop" of geoBASIC programs. With it, you can detect when the user clicks on an icon or selects a menu item, and branch to the appropriate subroutine to execute the user's choice. When the subroutine RETURNs, MAINLOOP takes over again and waits for the next selection. Before executing MAINLOOP, set up menus (see MENU) and icons (see ICON). Everything after MAINLOOP in a geoBASIC program should be subroutines which execute when a menu item is selected or an icon is clicked on.

Example:

```
10 ICON "NEW"
20 MENU "START"
30 MAINLOOP
40 END
100 PRINT "YOU PRESSED ICON 1":RETURN
200 PRINT "YOU PRESSED ICON 2":RETURN
```

This short program segment sets up the MENU and ICON, then goes into MAINLOOP and waits for the user to do something. Suppose that when you designed the icon list, you specified that the program should branch to line 100 if icon #1 was clicked on and to line 200 if icon #2 was clicked on. Clicking on icon #1 or #2 executes lines 100 or 200, printing the messages on the screen. The program then goes back to MAINLOOP to wait for the next time the user clicks on an icon.

MENU

FORMAT: MENU <string>

Places a menu at the top of the screen. The menu must have been designed previously using the Menu Editor. The <string> can be a string constant or string variable, and specifies the name of the Menu to place on the screen. This is the name used when you constructed the Menu using the Menu Editor. See MAINLOOP for an example of how your program should respond when the user selects one of the Menu items you have placed on the screen. Only the menu subitems present in the most recent MENU call are active.

Example:

```
10 MENU "MYMEN"
```

MID$

FORMAT: MID$(<string>,<expr 1>,[<expr 2>])

This function returns a sub-string taken from within a larger string given by <string>. <Expr 1> determines the starting position of the sub-string within the larger string. If <expr 1> is larger than the length of the <string>, then the null string is returned. < Expr 2> is optional and specifies how many characters are to be included in the sub-string, starting from the position determined by <expr 1>. If <expr 2> is left out, then the entire balance of the string is included in the sub-string. If <expr 2> is zero, then the null string is returned. If <expr 2> is larger than the length of the <string> from the starting position to the end of the string, then the rest of the string is returned. Both <expr 1> and <expr 2> can have values from 0 to 255.

Example:

```
10 A$="HELLO"
20 B$="THERE EVERYONE YOU"
30 PRINT A$+MID$(B$,7,8)
```

This sample program will print "HELLO EVERYONE".

MOUSE

FORMAT: MOUSE <expr>

This command turns the mouse on or off. The <expr> determines whether the mouse will be turned on or off. If it is equal to 0, then the mouse pointer is turned off. If the <expr> is any number other than 0, then the mouse pointer is turned on, making the mouse pointer visible on the screen.

Example:

```
10 MOUSE 1
```

MOUSEIN

FORMAT: MOUSEIN (<expr>,<expr>,<expr>,<expr>)

This function checks to see if the mouse is within the boundaries of the rectangle defined by the four numeric expressions. The first two expressions define the X and Y coordinates of the top left corner of the rectangle and the last two expressions define the bottom right corner of the rectangle. If the mouse is within the rectangle, then MOUSEIN returns TRUE (-1), and if the mouse is outside the rectangle, then MOUSEIN returns FALSE (0). The X coordinate expressions must evaluate to numbers between 0 and 319, while the Y coordinate expressions must evaluate to numbers between 0 and 199.

Example:

```
10 MW=MOUSEIN (10,20,50,100)
```

MOUSEX

FORMAT: MOUSEX (<expr>)

This function returns the current X coordinate of the mouse pointer. The <expr> can be any valid numeric expression, since it is not used. The value returned will be between 0 and 319.

Example:

```
10 X=MOUSEX(1)
```

MOUSEY

FORMAT: MOUSEY(<expr>)

This function returns the current Y coordinate of the mouse pointer. The <expr> can be any valid numeric expression, since it is not used. The value returned will be between 0 and 199.

Example:

```
10 Y=MOUSEY(1)
```

NEWPAGE

FORMAT: NEWPAGE

Advances the printer to the top of the next page.

Example:

```
10 NEWPAGE
```

NOT

FORMAT: NOT <expr>

The NOT operator can be used in two ways, just as the AND (see above) and the OR (see below) operators. First of all, NOT "complements" the value of each bit in the expression, which must evaluate to a number. The complement result of using NOT produces the expression increased by one and with a negative sign. Thus, NOT (45) results in –46. The second use of NOT is with expressions which are evaluated to be true or false. If an expression evaluates to be true, then NOT <expr> is false. If the expression evaluates to be false, then NOT <expr> is true.

Example:

```
10 AB=10:BA=20
20 IF NOT(AB=BA) THEN PRINT "NOT EQUAL!"
```

Since AB is not equal to BA, the expression (AB=BA) is false. Thus, NOT(AB=BA) is true, and the program will print "NOT EQUAL".

ON

FORMAT: ON <expr> GOTO/GOSUB <expr>[,<expr>]...

This statement allows your program to GOTO or GOSUB to one of the line numbers specified by the list of <expr> after GOTO or GOSUB, depending on the value of the first expression. The list of <expr> may be constants or labels. If the value of the first expression is 1, then the ON statement will GOTO or GOSUB to the first line number in the list. If the value of the expression is 2, then the ON statement will GOTO or GOSUB to the second line number in the list, and so on. If the value of the expression is not an integer, the fractional portion of it is ignored. If the value of the expression is zero or a number larger than the number of line numbers in the list, then the ON statement is ignored and execution continues with the next statement in the program. If the value of the expression is less than zero, an ?ILLEGAL QUANTITY error occurs. The ON statement can replace a whole series of IF statements for more efficient programs.

Example:

```
10 X=5
20 ON X-4 GOSUB 100,200,300,400:REM WILL GOSUB TO 100

10 ON EQ/10 GOTO 200,250,300,350,400,400,400
20 ON X-4 GOSUB @NM1,@NM2,@NM3
```

If there are values which the expression will never equal, you must still include a dummy line number in the list of line numbers if the variable COULD equal a value which is higher. You may also make the ON statement branch to the same line number for several different values of the expression.

ONERR

FORMAT: ONERR <expr>

Redirects errors to a line number. The geoBASIC stack is

cleared so you can't tell where you came from. Routine must end with a GOTO or MAINLOOP.

OPEN

FORMAT: OPEN <filename$>,[<expr>]

This command opens a channel for input or output to a file on the disk drive. The <filename> is a string constant or string variable specifying the filename for the disk file. The expression, if used, specifies which disk drive to load the file from. Valid disk drive numbers range from 8 to 11. If the expression is not used, then OPEN will try to open the file on the current disk drive.

Example:

```
10 OPEN "TEST"
```

OR

FORMAT: <expr> OR <expr>

The OR operator can be used for two purposes. As a mathematical (boolean) operator, it is used to combine two numbers together to produce a result. Each bit in the first expression is ORed against the corresponding bit in the second expression. The bit in the result is equal to 1 if the bit in either expression is 1. If the bit in both expressions is 0, then the bit in the result is zero. Thus, you get:

0 OR 0 = 0 1 OR 0 = 1 0 OR 1 = 1 1 OR 1 = 1

Each of the expressions must evaluate to a number between –32768 and +32767. If either of the expressions evaluate to a number outside this range, it will cause an ?ILLEGAL QUANTITY error.

Example:

```
10 X=32007 OR 28761: PRINT X
```

This would produce the result 32095. To see why, convert each of the numbers to binary: 32007 is 0111110100000111 and 28761 is 0111000001011001. ORing each bit of the two numbers:

```
0111110100000111 OR
0111000001011001
----------------------
0111110101011111 (binary)
or 32095 (decimal).
```

The other way to use the OR operator is to test the truth of two expressions. Each expression is generally an IF statement (see IF, above). The result evaluates as true if either of the expressions are true, and if both expressions are false, then the result is false. The "truth table" for the OR statement looks like:

First Expression	Second Expression	Result Expression
T	T	T
F	T	T
T	F	T
F	F	F

Example:

```
10 X=7:Y=10:Z=15
20 IF X=7 OR Y=10 THEN PRINT "TRUE"
30 IF X=7 OR Y=10 OR Z=15 THEN PRINT "TRUE"
40 IF X=5 OR Y=10 THEN PRINT "TRUE"
50 IF X=7 OR Y=12 THEN PRINT "TRUE"
60 IF X=5 OR Y=12 THEN PRINT "TRUE"
```

The statements in lines 20, 30,40 and 50 will print the word "TRUE" when you run this short program because at least one of the statements is true. The statement in line 60 will not, because

both expressions being tested are false. Note that in line 30, the OR statement is testing the truth of three statements (X=7, Y=10, and Z=15). This works by evaluating the statements two at a time. Thus, the truth of X=7 OR Y=10 is tested. In this case, this evaluates to be true. Then the result of this test (TRUE) is tested with the third statement, Z=15. Since Z is 15, this evaluates to: TRUE OR TRUE, which is TRUE. Larger groups of statements can be tested for truth in this way, and statements can be grouped together using parentheses:

10 IF (X=5 OR Y=10) OR (Z=20 OR Z*Y=200) THEN....

In this example, the truth of X=5 OR Y=10 is evaluated and stored. Then the truth of Z=20 OR Z*Y=200 is tested and stored. Finally, the two stored results are tested against each other for the final result. You can also combine OR statements with AND statements (see AND, above).

When a statement evaluates as FALSE, the value 0 is assigned to the result, and if the statement evaluates as true, the value of -1 is assigned to the result. Your program can determine the numerical value that the expression evaluates to by equating the expression to a variable:

Example:

```
10 X=10:Y=20
20 RES1=(X=20)
30 RES2=(X=10) OR (Y=20)
```

The variable RES1 will be zero, since the statement (X=20) is false. The variable RES2 will be -1, since both statements (X=10) and (Y=20) are true. As above, you can combine more than two expressions and evaluate the numerical result.

PATTERN

FORMAT: PATTERN <expr>

This command sets the pattern which will be used for all filled

shapes, such as RECT. The expression must evaluate to a number between 0 and 31.

Example:

```
10 PATTERN 10
```

PEEK

FORMAT: PEEK(<expr>)

This function returns the contents of the memory location specified by the expression which must evaluate to an integer between 0 and 65535. The contents will always be a number between 0 and 255.

Example:

```
10 A=PEEK(45)+256*PEEK(46)
20 PRINT PEEK(100)
```

POINT

FORMAT: POINT <expr>,<expr>

This command turns on the screen pixel located at the X and Y coordinates given by the two expressions. The X coordinate expression must evaluate to a number between 0 and 319, while the Y coordinate expression must evaluate to a number between 0 and 199.

Example:

```
10 POINT 10,20
```

POKE

FORMAT: POKE <location>,<expr>

The POKE command writes the one-byte value given by <expr> into the memory location given by <location>. The <location> expression must evaluate to an integer between 0 and 65535 and the expression must evaluate to an integer between 0 and 255. If either quantity is outside of its range, an ?ILLEGAL QUANTITY error will result.

The POKE command writes the expression directly into a memory location. Extreme care should be exercised when using this command, since putting the wrong value into a memory location could cause your computer to lock-up.

Example:

```
10 A=2040:B=0:POKE A,B
```

PRASCII

FORMAT: PRASCII [<expr>][<,/;><expr>]...

The PRASCII command is used to print data on the printer. The expressions can be of any type, including mathematical formulas, variables, strings and numeric or string constants. If no expressions are specified, then a blank line is printed.

The punctuation used between expressions determines how the items will be printed out. The 80-column line is broken up into 8 print zones of 10 characters each. If the expressions are separated by commas, the next value is printed at the beginning of the next zone. If the expressions are separated by semicolons (;), then the next value is printed immediately following the previous value. Using a semicolon at the end of a PRASCII statement suppresses the carriage return which would otherwise cause the next PRASCII statement to print at the beginning of the next line. Even

when a semicolon is used between expressions, numeric items are always followed by a space and positive numbers are preceded by a space.

You may also use a space or no separation at all between string variables or string constants which will have the same effect as using a semicolon.

Example:

```
10 X=10
20 PRASCII "X=";X;" X SQUARED IS ";X*X
30 A$="DAVID":B$="FRED":C$="LARRY"
40 PRASCII A$B$;C$,A$
```

The results of line 20 would print:

X=10 X SQUARED IS 100

and line 40 would print:

DAVIDFREDLARRY DAVID

PRINT

FORMAT: PRINT[<expr>][<,/;><expr>]...

The PRINT command is normally used to print data on the screen, although the output can be redirected to the printer using the PRNTER command. The expressions can be of any type, including mathematical formulas, variables, strings and numeric or string constants. If no expressions are specified, then a blank line is printed.

The punctuation used between expressions determine how the items will be printed out. The 80-column logical screen line is broken up into 8 print zones of 10 characters each. If the expressions are separated by commas, the next value is printed at the beginning of the next zone. If the expressions are separated by

semicolons (;), then the next value is printed immediately following the previous value. Using a semicolon at the end of a PRINT statement suppresses the carriage return which would otherwise cause the next PRINT statement to print at the beginning of the next line. Even when a semicolon is used between expressions, numeric items are always followed by a space and positive numbers are preceded by a space.

You may also use a space or no separation at all between string variables or string constants which will have the same effect as using a semicolon.

Example:

```
10 X=10
20 PRINT "X=";X;" X SQUARED IS ";X*X
30 A$="DAVID":B$="FRED":C$="LARRY"
40 PRINT A$B$;C$,A$
```

The results of line 20 would print something like:

X=10 X SQUARED IS 100

and line 40 would print:

DAVIDFREDLARRY DAVID

PRNTER

FORMAT: PRNTER <expr>

This function controls whether output of PRINT statements is routed to the printer or to the screen. If the expression is 0, then any subsequent PRINT statements will route output to the screen. If the <expr> is any other number, then output is routed to the printer instead of the screen.

Example:

```
10 PRNTER 0
```

PROCESS

FORMAT: PROCESS <expr>,<expr>

This command sets up a process which interrupts your main program and executes automatically on a periodic basis. The first expression defines the line number (or label) of the subroutine which will be executed. The second expression specifies the time (in 60ths of a second) before the process subroutine will be executed. The timer is reset after the process subroutine is executed. The subroutine must end with a RETURN. A maximum of 8 processes can be defined and running in a program at any one time. A running process is turned off by using the DELPROC command (see above).

Example:

```
10 PROCESS 10,100:REM EXECUTES THE SUBROUTINE
                      BEGINNING AT LINE 10 EVERY 100
                         JIFFIES.
```

PROMPT

FORMAT: PROMPT <expr>,<expr>,<expr>

Turns the GEOS prompt (line which flashes during INPUT) off or on. The first <expr> determines whether the prompt will be turned off or on. If it is equal to 0, then the prompt is turned off. If it is any other number, then the prompt is turned on. The prompt will be located at the X and Y coordinates given by the second and third expressions. The X coordinate expression must evaluate to a number between 0 and 319, and the Y coordinate expression must evaluate to a number between 0 and 199. The X and Y coordinate expressions are required even if you are turning the prompt off.

Example:

10 PROMPT 1, 10,20

PRSCREEN

FORMAT: PRSCREEN <expr>

This command prints a copy of whatever is on the graphics screen (where your program output is shown) on the printer. If the <expr> evaluates to zero, then the screen is printed as viewed. If the <expr> evaluates to 1, then the screen is rotated 90 degrees, with the top half of the screen being printed on the left side of the paper, and the bottom half of the screen being printed on the right side of the paper.

Example:

10 PRSCREEN 0

PTREC

FORMAT: PTREC <recordnum>

This command is used to point to a specific record in a VLIR file. The file must have been opened with CREATE, OPEN or DBFILE. Recordnum can be a number or numeric variable between 0 and 127. ALWAYS USE THIS COMMAND AFTER WRITING TO A RECORD! See the WRITE command.

RDBYTE

FORMAT: RDBYTE <variable$>[,<variable$>,...]

This command is used to fill variables with information from a disk file. The file being read from must have been previously opened with the OPEN, CREATE, or DBFILE commands. While numeric variables are permitted they are not recommended

because if the information coming in from the disk is not numeric an error will result. Use string variables and cast them to numbers with the VAL() command instead. This command differs from the DREAD command in that only single bytes are read from the file rather than strings. For example if the file contained:

now,is,the,time

DREAD a$,b$,c$,d$ would return "now", "is", "the", "time" while RDBYTE a$,b$,c$,d$ would return "n", "o", "w", "," (see also DREAD).

READ

FORMAT: READ <variable>[,<variable>]...

The READ statement is used to fill variables with values from information contained in DATA statements. The type of the data which is read from the DATA statements must match the type of the variable in the READ statement or the error ?SYNTAX ERROR will occur.

DATA statements are accessed in order by the READ statements, and a READ statement can cross from one DATA statement to another. Each subsequent READ statement picks up where the last one left off (see also the RESTORE statement, below). If there are not enough data items in DATA statements for the variables in the READ statements, you will get the ?OUT OF DATA error.

Example:

```
10 READ A,B,C$
20 10,20,HELLO
```

RECT

FORMAT: RECT <expr>,<expr>,<expr>,<expr>

This command draws a filled rectangle on the screen. The first two expressions specify the X and Y coordinates of the upper left corner of the rectangle, and the last two expressions select the X and Y coordinates of the lower right corner of the rectangle. The X coordinate expressions must evaluate to a number between 0 and 319, while the Y coordinate expressions must evaluate to a number between 0 and 199. The rectangle is drawn using the pattern chosen using PATTERN.

Example:

```
10 PATTERN 10
20 RECT 10,20,20,30
```

REDRAW

FORMAT: REDRAW <expr>

If a desk accessory is run while your program is running, it will likely mess up the graphics of your program's screen. Menus and icons are also deactivated by a desk accessory. When the desk accessory is closed up, if the REDRAW command is present in your program and has been executed, then it goes to the line number given by the expression. This section can be used to redraw the screen, reset menus and icons and any other functions necessary to get your program going again.

Example:

```
10 REDRAW 100
.
.
100 MENU "STUFF":CLS:PRINT "HELLO THERE":MAINLOOP
```

REM

FORMAT: REM [<text>]

The REM statement and anything which follows it on the line are ignored by your program. Thus, you can put anything you like on the REM line, although this command is typically used to put comments and notes in your program to make it easier to understand.

Example:

```
10 PRINT "HELLO WORLD"
20 REM SENDS THE MESSAGE "HELLO WORLD" TO THE
   PRINTER
```

REPEAT...UNTIL

FORMAT: REPEAT...UNTIL <expr>

This command establishes a loop structure. All commands following REPEAT will be executed until the command UNTIL is reached. If the expression following UNTIL is FALSE (0), then the program loops back to REPEAT and executes the block of statements between REPEAT and UNTIL again. This continues until the expression following UNTIL is TRUE (-1). The program then leaves the loop and continues execution with the statement immediately following UNTIL. Note that the block of statements between REPEAT and UNTIL will always be executed at least once, since the test of the condition doesn't occur till the end of the loop. If the expression following UNTIL is never true, then the loop will continue indefinitely.

Example:

```
10 A=10
20 REPEAT:A=A+1:PRINT A
30 UNTIL A=20:REM WILL EXECUTE 10 TIMES
```

RESTORE

FORMAT: RESTORE

When using READ statements to read data stored in DATA statements, the READ statements read the data beginning with the first DATA statement in the program and proceeding through the DATA statements in numerical order. The RESTORE statement causes the next READ statement to begin reading data from the first DATA statement in the program again. Thus, information in DATA statements can be used many times to put values in the variables in READ statements.

Example:

```
10 FOR LP=1 TO 10: READ AB(X):NEXT LP
20 RESTORE
30 FOR LP=1 TO 10 READ CD(X):NEXT LP
40 DATA 1,2,3,4,5,6,7,8,9,0
```

This sample program fills the two arrays with the same data.

RIGHT$

FORMAT: RIGHT$(<string>,<expr>)

This function returns the number of characters specified by <expr> (which must evaluate to a number between 0 and 255) from the right end of the string specified by <string>. If the value of the expression is 0, then the null string is returned. If the value of the expression is bigger than the length of <string>, then all of <string> is returned.

Example:

```
10 A$="BERKELEY SOFTWORKS"
20 B$=RIGHT$(A$,4):PRINT B$:REM WOULD PRINT "WORKS"
```

RND

FORMAT: RND(<expr>)

The RND function returns a random number between 0 and the value of the expression minus 1. The number is a floating point number, and it may be negative.

Example:

```
10 A=RND(5):PRINT A
20 A=INT(RND(10))+1:REM RANDOM INTEGER BETWEEN 1 AND 10
```

RUN

FORMAT: RUN[<line-number>]

RUN is used to start execution of the program currently in memory. The optional <line-number> will cause the program to start to RUN at the specified line number. Otherwise, the RUN command begins the program execution at the first line of the program. If the <line-number> doesn't exist, the error UNDEF'D STATEMENT will occur.

A running program stops and returns to your control when the END or STOP statements are encountered, when the last line of the program is reached, when a BASIC error occurs, or when you press the [RUN/STOP] key.

Example:

RUN	**starts at the first line of the program**
RUN 200	**starts the run at line 200**

SAVE

FORMAT: SAVE <filename>,<expr>,<expr>,[<expr>]

This statement writes the contents of memory to a file on disk. The filename identifies the name of the file you want to write. The filename may be contained in a string variable.

The first expression is the memory address where the file starts. It must evaluate to a number between 0 and 65535. The second expression is how much of the file (how many bytes) to save. The third optional expression specifies the device number that the file is to be saved to . It must evaluate to a number between 8 (first disk drive) and 11. If the expression is left out, the current drive will be used.

Example:

SAVE "PIC",40960,8000 **(saves the graphics screen to disk.)**
SAVE B$,32768,200,8 **saves the filename given by B$ to disk. The file will be saved from memory location 32768 and 200 bytes of it will be saved.**

SETCOL

FORMAT: SETCOL <expr>

This command sets the variety of color and pattern parameters, depending on the expression used.

1) For lines, (see LINE, above) if the expression evaluates to 0, the line is drawn in background color, and if the expression is not zero, the line is drawn in foreground color.

2) For rectangles the low four bits (nibble) of the expression sets the background color (used for COLRECT) and the high nibble sets the foreground color (used for RECT). To use a foreground color of 5 and a background color of 8, you would use SETCOL 8+16*5.

3) For a framed rectangle, created with FRECT, the expression sets what the four lines which make up the frame of the rectangle

will look like. Instead of setting the color of the lines, SETCOL sets the line pattern. The 8 bits which make up the number passed to SETCOL can be either On or Off. If the bit is On, then it shows up in the line, and if the bit is Off, then that is also off in the line. For example, the number 255 (binary 11111111) has all bits on, so the line drawn with FRECT will be solid. 85 (binary 10101010) will produce a dashed line, with every other pixel of the line being On.

Example:

```
10 SETCOL 80
20 COLRECT 10,10,20,20
30 FRECT 40,40,80,80
```

This short program puts up a green colored rectangle and a dotted, framed rectangle (80 is 01010000 in binary).

SETPOS

FORMAT: SETPOS <X expr>,<Y expr>

This command sets the cursor to the position where the PRINT command will output the next character. The X expression must evaluate to a number between 0 and 319 and the Y expression must be between 0 and 199.

Example:

```
10 SETPOS 100,100
```

SGN

FORMAT: SGN(<expr>)

The results of using the SGN function on an expression is to return the following depending on the value of the expression: –1 if the expression is less than zero, 0 if the expression is equal to zero, and 1 if the expression is greater than 0.

Example:

```
10 X=2:PRINT SGN(X):REM PRINTS 1
20 X=0:PRINT SGN(X):REM PRINTS 0
30 X=2:PRINT SGN(X*(-1)):REM PRINTS -1
```

SIN

FORMAT: SIN(<expr>)

This function returns the sine of the expression which must evaluate to a numeric value. The angle is in radians.

Example:

```
10 PRINT SIN(4):REM PRINTS -.756802495
```

SOUND

FORMAT: SOUND <expr>,<expr>,<expr>,<expr>

This command generates sound. The key to creating a sound is the shape of the envelope. A sound rises to a maximum volume from zero (attack), falls back to a lower volume (decay), holds that volume for awhile (sustain) and then drops back to zero (release). The period of time which is spent on each part of the curve defines how the sound will behave. If you look at a plot of the sound of a piano versus, say, the sound of a trumpet, you can see the difference in their sound envelopes. It is this difference which causes the notes produced by each instrument to sound different to your ear.

The first expression in SOUND specifies which voice will be used for the sound and it must evaluate to a number from 0 to 3. The second expression is the frequency of the note to play. In the third expression, the low nibble specifies the attack time and the high nibble specifies the decay time. In the fourth expression, the low nibble specifies the sustain and the high nibble specifies the release time. Both the third and fourth expressions must evaluate to

numbers between 0 and 255. Since both nibbles of the third and fourth expression are used to define separate items, the attack, decay, sustain and release can each have only 16 different values. The amount of time corresponding to each of the 16 values depends on whether you are using attack or decay/release. They are:

Value	Attack time	Decay/Release time
0	2 ms	6 ms
1	8 ms	24 ms
2	16 ms	48 ms
3	24 ms	72 ms
4	38 ms	114 ms
5	56 ms	168 ms
6	68 ms	204 ms
7	80 ms	240 ms
8	100 ms	300 ms
9	250 ms	750 ms
10	500 ms	1.5 s
11	800 ms	2.4 s
12	1 s	3 s
13	3 s	9 s
14	5 s	15 s
15	8 s	24 s

The Sustain is not controlled in this way. Although it also can have a value from 0 to 15, this value defines the proportion of the peak volume (at the end of the attack) that sustain will be.

Example:

```
10 SOUND 1,10000,5*16+10,8*16+3
```

By multiplying the Decay by 16 and adding the Attack, the two nibble values can be constructed into a single number to use with SOUND. The Release and Sustain (fourth expression) also works this way.

SPC

FORMAT: SPC(<expr>)

This function will (when used with the PRINT statement) print the number of spaces specified by the expression, which must evaluate to a positive integer between 0 and 255. When sending spaces to a printer using SPC, if the end of the line is encountered, a carriage and line feed will be sent, and no further spaces will be output on the next line. NOTE: On the screen a single space is five pixels.

Example:

```
10 PRINT "LEFT AND";
20 PRINT SPC(20); "RIGHT";
```

This example will print:

LEFT AND RIGHT

SPRCOL

FORMAT: SPRCOL <expr>,<expr>

Sprites may be up to three colors, but only one of those colors can be different for each sprite. The other two colors are the same for all the sprites. These colors are set by using this command. The two expressions are the color numbers of the two colors to set. There are sixteen colors (numbered 0 to 15). They are: black, white, red, cyan, purple, green, blue, yellow, orange, brown, light red, dark grey, grey, light green, light blue, and light grey.

Example:

```
10 SPRCOL 2,5
```

SPRITE

FORMAT: SPRITE <string>

Puts sprites on the screen. The sprites must have been designed previously using the Sprite Editor. The <string> is the name of the Sprite definition file you want to use. It must have the same name you gave the file when you designed it using the Sprite Editor. If a velocity, sprite linking or animation was specified when the sprites were designed, using this command will cause the sprites to begin to move, animate, etc. The sprites will appear initially at the X and Y locations specified when they were designed.

Example:

```
10 SPRITE "SNEW"
```

SPRT

FORMAT: SPRT <sprite # (3-8)>,<sprt attribute #>,<new value or variable>

This allows you to change or retrieve sprite information while the sprite is moving, etc. This command is a bit different from the others—basically you specify which sprite (3-8) you'd like information about or change its information. You then pass which attribute you'd like to get info about or change—attribute numbers range from 0-5:

0 color
1 X velocity (or X position of TRAIL's end if TRAIL was used)
2 Y velocity (or Y position of TRAIL's end if TRAIL was used)
3 X position (0-512)
4 Y position (0-256)
5 timeout time

Therefore, if you'd like an accelerating sprite you might add something like this in your code:

Example:

```
500 FOR XVEL = 50 TO 200 STEP 10
510     SPRT 3, 1, XVEL : REM SET VELOCITY IN SPRITE 3
520 NEXT XVEL
```

If you pass a negative sprite attribute number, then a variable is assigned the value of the attribute. Notice that there is no "negative zero" so a sprite color cannot be returned in a variable—the color of a sprite should never change unless specifically told to do so, but velocities and positions may change as the sprite moves about. For example, if you'd like to know where a sprite is you could do:

```
600 SPRT 3, -3, xPos : SPRT 3, -4, yPos
```

xPos and yPos would then be assigned the current position of the sprite.

SQR

FORMAT: SQR(<expr>)

This function returns the square root of the expression, which must evaluate to a positive number. If it is negative, an ?ILLEGAL QUANTITY error will occur.

Example:

```
10 FOR X=1 TO 10
20 PRINT SQR(X)
30 NEXT X
```

STR$

FORMAT: STR$(<expr>)

This function converts the numeric value given by the expression to a string. Any number in the expression, when converted to a string, will be followed by a space. If it is a positive

number, it will also be preceded by a space. Converting a number to a string is handy for formatting reports.

Example:

```
10 PRINT STR$(3450)
20 X=10:Y=20:A$=STR$(X*Y)
```

SYSINFO

FORMAT: SYSINFO <PARAMETER NUMBER>, <VARIABLE>

SYSINFO will return some system relevant stuff in case you'd like to have your program easily portable on other machines. The parameter number is from 0 to 16 and is returned in the variable.

Parameter #	Description
0	Machine type—0 = C64, 1 = C128, 2 = APPLE (future use)
1	GEOS Kernal version
2	Language (0 = English)
3	Current drive (8,9)
4	Maximum right edge of graphics screen (319 on C64)
5	Maximum bottom edge of graphics screen (199 on C64)
6	Last basic error encountered (see Errors list)
7	Current hour
8	Current minutes
9	Current seconds
10	Current month
11	Current day
12	Current year
13	Calling option (0 - none, 1 - double clicked, 2 - print)
14	Name of file double clicked on (if any)
15	Current time (hour : minutes : seconds)
16	Current date (month / day / year)

The variables for parameters 0 - 13 should be floating point

whereas the last three (14 - 16) should be string variables. For example, if you'd like to see what error occurred in your program with the ONERR statement you could do something like this:

Example:

```
10 BREAK = 47 : ONERR 1000
.
.
.
.
1000 SYSINFO 6, CHECKBREAK
1010 IF CHECKBREAK = BREAK THEN PRINT "HAHA YOU CAN'T
     GET OUT!!":GOTO 10
```

TAB

FORMAT: TAB(<expr>)

The TAB function moves the cursor to the position on the line specified by expression, which must evaluate to a number between 0 and 7. There are tab positions every 40 pixels, for a total of 8 on a line. If the current cursor position is to the right of the expression, then the cursor moves to the new position. If the current position is equal to or already past the position specified by the expression, then TAB has no effect. TAB must be used with PRINT to have any effect.

Example:

```
10 PRINT "FIRST";TAB(3);"LAST"
```

TAN

FORMAT: TAN(<expr>)

This function returns the tangent of the value given by the expression in radians. The TAN function is undefined at certain

points. If you choose a point at which the TAN function is undefined, a ?DIVISION BY ZERO error will result.

Example:

```
10 X=1:Y=TAN(X*2):PRINT Y
```

This example will print –2.18503987.

TESTPT

FORMAT: TESTPT (<expr>,<expr>)

This function checks to see if a point on the screen is turned on, that is, not the background color. The first expression is the X coordinate of the point, and the second expression is the Y coordinate. The X coordinate expression must evaluate to a number between 0 and 319, and the Y coordinate expression must evaluate to a number between 0 and 199. If the point on the screen is on, TESTPT returns TRUE (–1), and if the point is not on, TESTPT returns FALSE (0).

Example:

```
10 TESTPT (20,30)
```

USR

FORMAT: USR(<expr>)

This function causes a BASIC program to begin executing a machine language subroutine which has its starting address pointed to by the contents of two memory locations: 785 (low byte) and 786 (high byte). This starting address must be inserted into these memory locations before making the USR call by using the POKE statement (see above).

The value which the expression evaluates to is stored in the floating point accumulator starting at memory location 97, which

can be accessed by the Assembler code. The result of the USR subroutine is stored in these same locations when the subroutine returns to the current BASIC program. The USR machine language subroutine should end with RTS so that execution continues with the statement immediately following the USR call in the BASIC program.

Example:

```
10 POKE 786,HIBYTE:POKE 785,LOBYTE
20 USR(X)
```

VAL

FORMAT: VAL(<string>)

This function converts the string specified by <string> to a numeric VALue. If the first non-blank character is not a number or a sign (+ or -), then the function returns 0. The function terminates and returns a value either when the end of the string is encountered or any non-digit character is found, except that the decimal point or exponential e are allowed.

Example:

```
10 A$="4567":B$="+1.44e04":C$="X23"
20 A=VAL(A$):B=VAL(B$):C=VAL(C$)
30 PRINT A,B,C
```

This example program would print: 4567 1.44e04 0

VOICE

FORMAT: VOICE <expr>,<expr>

This command sets the type of instrument which will be played when the SOUND command is used for each of the four available voices. The first expression specifies the voice number, and can range from 0 to 3. The second expression specifies the instrument

for that voice. Instrument numbers are :

0 - noise
1 - woodwind
2 - strings
3 - brass
4 - bell

Example:

```
10 VOICE 1,2
```

WHILE...LOOP

FORMAT: WHILE <expr> ... LOOP

This command establishes a loop structure. If the expression following WHILE is TRUE(1), then the program executes the block of statements between WHILE and LOOP. When the statement LOOP is reached, the program returns to the WHILE statement and executes the block of statements again. This continues until the expression following WHILE is FALSE (0). The program then leaves the loop and continues execution with the statement immediately following LOOP. Note that the block of statements between WHILE and LOOP may never be executed since the test of the condition occurs at the beginning of the loop. If the expression following WHILE is always true, then the loop will continue indefinitely.

Example:

```
10 X=10
20 WHILE X<20:X=X+1:PRINT X
30 LOOP
```

WINDOW

FORMAT: WINDOW <expr>,<expr>,<expr>,<expr>

This command defines a print window on the screen. All PRINT command output will be confined to the window. The first two arguments are the X and Y coordinates of the top left corner of the window, and the last two arguments are the X and Y coordinates of the bottom right corner of the window. The X coordinate expressions must evaluate to a number between 0 and 312 (rounded down to the nearest 8th pixel), and the Y coordinate expressions must evaluate to a number between 0 and 199. It is possible to have more than one print window going at once, switching the cursor between them under program control:

Example:

```
10 WINDOW 10,10,100,100 :REM DEFINE THE 1ST WINDOW
20 PRINT "HELLO WORLD"
30 PRINT "NEXT LINE OF WIND 1"
40 X(1)=XPOS(0):Y(1)=YPOS(0) :REM SAVE CURSOR POS.
50 WINDOW 20,20,200,200 :REM DEFINE THE 2ND WINDOW
60 PRINT "HELLO AGAIN"
70 PRINT "NEXT LINE OF WIND 2"
80 X(2)=XPOS(0):Y(2)=YPOS(0) :REM SAVE CURSOR POS.
90 SETPOS X(1),Y(1) :REM CURSOR BACK TO WINDOW 1
100 WINDOW 10,10,100,100 :REM REDEFINE WINDOW 1
110 PRINT "THIS IS AGAIN IN WINDOW 1"
```

WRITE

FORMAT: WRITE <var/var$/string>[,<var/var$/string>,...]

The WRITE command is used to write data to a disk file. The file must have been previously opened with the OPEN, CREATE or DBFILE commands. The data may be any type of variable, quoted strings, or numbers. ALWAYS USE THE PTREC COMMAND AFTER WRITING TO A RECORD! The information will not be written out to the disk until a PTREC command is issued. It does not matter what record number you point to. If you CLOSE the file (or try reading) without using PTREC the file will not be updated!

Example:

```
100 OPEN "MYFILE"
110 PTREC 0
120 WRITE A$,C,123,C%,D$(5),"HI THERE!"
130 PTREC 0
140 CLOSE
```

XPOS

FORMAT: XPOS (<expr>)

This function returns the X coordinate of the current position of the cursor (where PRINT statements produce output on the screen). The expression may be any number or variable. See WINDOW for a more extensive example of how to use XPOS.

Example:

```
10 A=XPOS(1)
```

YPOS

FORMAT: YPOS (<expr>)

This function returns the Y coordinate of the current position of the cursor (where PRINT statements produce output on the screen). The expression may be any number or variable. See WINDOW for a more extensive example of how to use YPOS.

Example:

```
10 A=YPOS(1)
```

Chapter 6—Disk and File Programming

Disk programming is usually considered difficult, and even though geoBASIC has a wealth of commands for the BASIC programmer, it can still be a challenge to many beginners. So this chapter brings all the disk and file commands under one heading, with easy to understand examples. For information on other geoBASIC commands please refer to the chapter on geoBASIC commands and their syntax.

CREATE <filename$>[,<drivenum>]

This command creates a VLIR file on the disk. A VLIR file is composed of a collection of records, each of which may have a maximum size of 32K bytes. There may be up to 128 of these records in the file (numbered 0 to 127). The filename parameter may be a string variable or quoted string of any length but only the first 16 characters are significant. Drivenum is an optional parameter and specifies the device number to create the file on. This number can range from 8 to 11. If this parameter is omitted the current drive will be used.

Unless changed with the HEADER command, CREATE will create only files of type 'BASIC DATA', with an empty permanent name string. DO NOT USE CREATE IF THE THIRD PARAMETER OF THE HEADER COMMAND IS ZERO! CREATE will not create sequential files properly!

Note that CREATE will leave the file in an open state so there is no need to issue an OPEN command prior to accessing the file.

Example:

```
10 CREATE "TEST" : REM CREATES 'TEST' ON THE CURRENT
   DRIVE
10 CREATE A$,9 : REM CREATES FILE NAMED IN A$ ON DEVICE #9
```

OPEN <filename$>,[<expr>]

This command opens a channel for input or output to a file on the disk drive. The <filename> is a string constant or string variable specifying the filename for the disk file. The expression, if used, specifies which disk drive to load the file from. Valid disk drive numbers range from 8 to 11. If the expression is not used, then OPEN will try to open the file on the current disk drive.

Example:

```
10 OPEN "TEST"
```

CLOSE

This statement closes the data file which was opened using the OPEN command. Only a single file can be OPEN at once, so you must use CLOSE to close any open file before OPENing another file.

Example:

```
10 CLOSE
```

HEADER <expr>,<string>[,<expr>]

The HEADER command allows you to specify the type of file and permanent file name of the files which will appear when using DBFILE or that will be created using CREATE. If the HEADER command is not used, then only files matching type BASIC DATA and with a permanent file name of "" (null string) will appear when using DBFILE or be created when using CREATE. The first <expr> must evaluate to a valid file type. Valid file types are:

BASIC (1), ASSEMBLY (2), DATA (3), SYSTEM (4), DESK_ACC (5), APPLICATION (6), APPL_DATA (7), FONT (8), PRINTER (9), INPUT_DEVICE (10), DISK_DEVICE (11), SYSTEM_BOOT (12), TEMPORARY (13), AUTO_EXEC (14), and NUM_FILE_TYPES (15).

Most of these file types would never need to be used or accessed by a user. For more information on the different file types, see page 398 of the GEOS Programmers Reference Guide.

The <string> is the permanent file name to match. If used, then only files which match this permanent file name will appear in the dialog box. The permanent file name is used by GEOS to help identify what application a file belongs to. For example, it is by the permanent file name that GEOS can tell the difference between GeoPaint and GeoWrite files. The permanent file name of any file can be seen from the desktop. Click on the file you are interested in to highlight it, then drop down the File submenu and click on Info. The name which appears in the Info box next to Class is the permanent file name.

The optional last <expr> is the file structure. It must evaluate to either 0 (sequential) or 1 (VLIR).

Example:

```
10 HEADER 3,"",1
```

PTREC <recordnum>

This command is used to point to a specific record in a VLIR file. The file must have been opened with CREATE, OPEN or DBFILE. Recordnum can be a number or numeric variable between 0 and 127. ALWAYS USE THIS COMMAND AFTER WRITING TO A RECORD! See the WRITE command.

APPEND <recordnum>

This command adds a new record to a VLIR file. Recordnum is either a number or a numeric variable that points to the record that will be appended to. For example:

before: **0 1 2 3 4 5 ...**

after an APPEND 2: **0 1 2 3 4 5 6 ...**
^
—- new record

All records after the appended record are moved up one record. If the last record would exceed 127 then an OUT_OF_RECORDS error will occur. There is a bug in this command that prevents APPENDing to record 126. APPEND will do an implicit PTREC to the new record (see also INSERT).

INSERT <recordnum>

This command adds a new record to a VLIR file. Recordnum is either a number or a numeric variable that points to where record will be inserted. For example:

before: **0 1 2 3 4 5 ...**

after an INSERT 2: **0 1 2 3 4 5 6 ...**
^
—- new record

All records after the inserted record are moved up one record. If the last record would exceed 127 then an OUT_OF_RECORDS error will occur. There is a bug in this command that prevents INSERTing to record 126. INSERT will do an implicit PTREC to the new record (see also APPEND).

DREAD <variable$>[,<variable$>,...]

This command is used to fill variables with information from a disk file. The file being read from must have been previously opened with the OPEN, CREATE, or DBFILE commands. While numeric variables are permitted they are not recommended because if the information coming in from the disk is not numeric

an error will result. Use string variables and cast them to numbers with the VAL() command instead. Each string in the file must be terminated with a carriage return or a comma. An error will occur if the string is longer than 255 characters or if you try to read from an empty record (see also RDBYTE).

RDBYTE <variable$>[,<variable$>,...]

This command is used to fill variables with information from a disk file. The file being read from must have been previously opened with the OPEN, CREATE, or DBFILE commands. While numeric variables are permitted they are not recommended because if the information coming in from the disk is not numeric an error will result. Use string variables and cast them to numbers with the VAL() command instead. This command differs from the DREAD command in that only single bytes are read from the file rather than strings. For example if the file contained:

now,is,the,time

DREAD a$,b$,c$,d$ would return "now", "is", "the", "time" while RDBYTE a$,b$,c$,d$ would return "n", "o", "w", "," (see also DREAD).

WRITE <var/var$/string>[,<var/var$/string>,...]

The WRITE command is used to write data to a disk file. The file must have been previously opened with the OPEN, CREATE or DBFILE commands. The data may be any type of variable, quoted strings, or numbers. ALWAYS USE THE PTREC COMMAND AFTER WRITING TO A RECORD! The information will not be written out to the disk until a PTREC command is issued. It does not matter what record number you point to. If you CLOSE the file (or try reading) without using PTREC the file will not be updated!

Example:

```
100 OPEN "MYFILE"
110 PTREC 0
120 WRITE A$,C,123,C%,D$(5),"HI THERE!"
130 PTREC 0
140 CLOSE
```

EOF(0)

This function is used to signal the end of a disk record. The argument may be any variable or a number. If the last DREAD or RDBYTE returned the last character of the record, or if PTREC is used on an empty record, then a TRUE (–1) is returned. Otherwise, EOF(0) returns zero.

Remember that there is a difference between an empty record and an unused record. An empty record is one that was created with INSERT or APPEND but nothing was written to it. An unused record has never been accessed at all. PTREC will generate an error if you try to point to an unused record, preventing the use of EOF()!

Example:

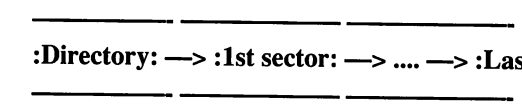
```
10 OPEN "MYFILE"
20 PTREC 0
30 WHILE NOT EOF(0)
40 RDBYTE A$
50 A = ASC(A$)+CHR$(0)
60 IF A > 31 OR A = 13 THEN PRINT A$;
70 LOOP
80 CLOSE
90 END
```

VLIR File Description

A VLIR (Variable Length Index Record) file is simply a

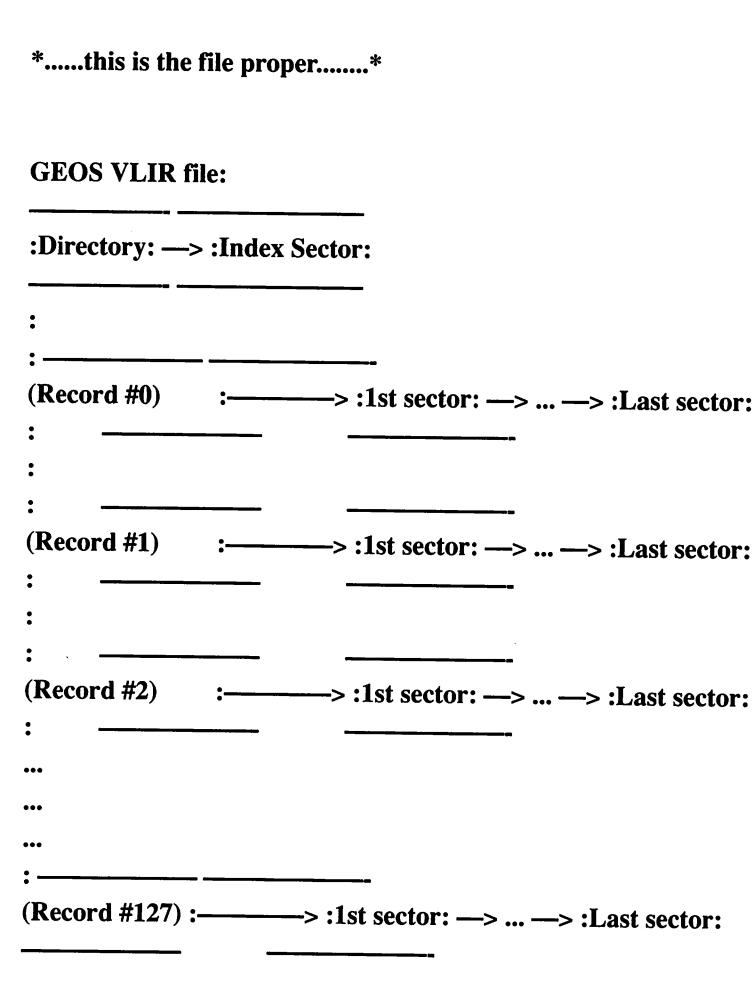
collection of 'records' that are linked together by a common sector. This 'index sector' contains pointers to each record on the disk. Think of each record as a file that is referenced by a number instead of a filename.

Normal Commodore file:



.....this is the file proper.....

GEOS VLIR file:



Chapter 7—The geoBASIC Utility Programs

THE MENU EDITOR

The menus located at the top of the screen are an integral part of a geoBASIC program and the GEOS environment. The words which are visible on the top line of the screen are called submenus. Moving the mouse pointer up to a submenu and clicking the left button causes the submenu to "drop down," making the items in the submenu visible on the screen. To select one of the menu items, move the mouse pointer to the item you want and click the left button. The submenu will then close up and your choice will be acted on. If you change your mind about wanting to use one of the menu items, simply move the mouse pointer away from the dropped-down submenu, and it will close up again.

The first step to using menus in programs of your own is to construct the menus with the Menu editor, which is built into your geoBASIC package. There are two major steps in building your own menus. The first is to specify the text which is to appear in each submenu and all its items. The second is to tie each menu item to a particular subroutine. When the menu item is selected while your program is running, the program will automatically branch to the subroutine at the line number which was specified when the menu was designed. Since geoBASIC automatically monitors menu selections for you, it is very easy to use your custom menus in your own programs, as we will see below.

STARTING THE MENU EDITOR

To start using the Menu editor from geoBASIC move the mouse pointer up to the Utilities menu and click the left button. Move the mouse pointer to the menu item menu and click the left button again. The first screen of the menu editor will appear. In the box in the center will be listed any menu files which exist on the current disk in the drive. You may select one of these files to edit

by clicking on it or type in the name of a new file under the prompt Create or edit an item Name:. Menu names can be a maximum of 5 characters long. To continue, press [RETURN] or click on OK. To return to geoBASIC just click on CANCEL.

USING THE MENU EDITOR

The next screen which appears is the menu editing screen. If you are creating a new menu, this screen appears with four submenus, each one containing four items. The number of submenus and items can be easily changed. If you are editing an existing menu, the condition of the menu when you last saved it to disk will be reflected on this screen. Let's take a look at the elements of the screen, one by one.

On the top line of the screen, you can see the menu as you actually build it. Because of the highly interactive nature of the Menu editor, it is easy to make changes to the menu and see immediately what they look like. The first submenu is GEOS. This is always the first submenu on the line, and you cannot remove it. The rest of the submenus on the line can be modified.

At the top of the menu edit box is a line which reads: Number of submenus. Alongside this line is a number, with an up arrow on the left and a down arrow on the right. To increase the number of submenus, click on the up arrow. To decrease the number of submenus, click on the down arrow. The number in the box will change to reflect the new number of submenus, and the submenus at the top of the screen will also change as submenus are added or removed. The maximum number of submenus is 8. Clicking on the up arrow when there are eight submenus selected will return to just a single submenu (GEOS).

Below the line where you set the number of submenus are the lines where you set the text which will appear in each submenu. There is one line for each submenu you specified above. The submenus are created with a default name of menu followed by a number. To edit the submenu name, click the mouse pointer on the

line for the submenu you wish to edit to move the cursor to that line and type in your changes. To delete a character, use the [DEL] key. Once you are done making changes on a particular line, click the mouse pointer on a different line or press [RETURN] to move the cursor to the next line down. The text changes to the submenu text will be reflected in the menus at the top of the screen.

Note: If you specify submenu names which are too long to fit in the menus at the top of the screen, the last few characters may be cut off and will not be visible. You are allowed 37 characters in the menus at the top of the screen. This maximum length limits the amount of text AND the number of submenus you can use. For example, if you use long submenu titles, only three or four may fit within the 37 character limit.

To the right of the submenu text are boxes (one for each submenu) for specifying how many items will be present in each submenu. There are initially four items in each submenu, but you can change that. The submenus DO NOT all have to have the same number of items. Each item box has an up arrow and a down arrow alongside the number of items. To increase the number of items in a given submenu, click on the up arrow. To decrease the number of items in a given submenu, click on the down arrow. The maximum number of items in a submenu is 12. Clicking on the up arrow when Items is 12 causes the number of items to return to 1.

To set the text of an item, move the mouse pointer to the submenu that the item belongs to and click the left button, causing the submenu to drop down. Then move the mouse pointer to the item you want to change and click the left button again. A dialog box will be presented on the screen for you to type in the next text of the item. Each item is created with the default nameaction followed by a number. To delete text, use the [DEL] key. When you are done changing the text, press [RETURN]. If you want to discard the changes, click on the button marked Cancel. The maximum length of text for an item is 24 characters.

The next dialog box will ask the question Gosub where if

selected? Here you enter the line number or label where the program should branch if the item you are currently working on is selected while the program is running. You may leave this blank, in which case nothing will happen if that submenu item is selected. This allows you to come back and fill in the subroutine when you have defined it in your geoBASIC program. If you changed your selection of where to Gosub and then change your mind, click on the button marked Cancel.

There is one menu item under the GEOS submenu. The text of the item and the subroutine to branch to if the item is selected are set up exactly the same way as the other submenu items. Often, the subroutine which is branched to by clicking on this item will put up a dialog box on the screen, identifying the program's author (that would be you!) and the program revision number.

When you have completed designing or changing your menu, click on OK. The menu file will automatically be saved to disk under the name you gave it earlier. If you decide not to store the changes or the new menu file, click on Cancel.

USING A MENU IN YOUR OWN PROGRAM

Using your new menu in your own program couldn't be simpler. To install the menu at the top of the screen, use the command:

```
10 MENU "MNAME"
```

where "mname" is the name you gave the menu when you created it. You then use the MAINLOOP command to wait for a menu item to be clicked on (among other things). If a menu item is clicked on, the program will branch automatically to the subroutine you specified when you built the menu.

THE BITMAP UTILITY

Bitmaps are small pictures which can be used in various ways in your own programs. A bitmap can be any size from 8 pixels

wide and 1 pixel high all the way up to 48 pixels wide by 42 pixels high. Once you have created a bitmap using the bitmap utility and stored the bitmap on disk, you can use the created bitmap as an icon in a Dialog box or as an icon in an Icon list (see below). Bitmaps can only be one color.

STARTING THE BITMAP UTILITY

To start using the Bitmap editor from geoBASIC move the mouse pointer up to the Utilities menu and click the left button. Move the mouse pointer to the menu item Bitmap and click the left button again. The first screen of the Bitmap editor will appear. In the box in the center will be listed any Bitmap files which exist on the current disk in the drive. You may select one of these files to edit by clicking on it or type in the name of a new file under the prompt Create or edit an item Name:. Bitmap names may be a maximum of 5 letters long. To continue, press [RETURN] or click on OK. To return to geoBASIC just click on CANCEL.

USING THE BITMAP UTILITY

The Bitmap utility screen consists of two main parts. On the left side of the screen is the working area, where you draw your bitmap in a magnified mode. On the top of the right side is a blank area which will show what your bitmap looks like in actual size. Also on the right side are two numbers, labeled Width and Height. These define the overall size of the bitmap you are creating. On the left side of each number is an up arrow, and on the right side is a down arrow. To increase the height or width, move the mouse pointer to the up arrow and click the left button. The Height can be any size between 1 and 42 pixels, but the width changes only in increments of 8 pixels, up to a maximum of 48 pixels wide. To decrease the height or width, click on the down arrow. As you adjust the dimensions, the bitmap frame on the left side of the screen will change to reflect the new size. At certain combinations of height and width, the size of the magnified bitmap frame may suddenly change drastically. There are two sizes of "dot" you can draw with when creating your bitmap, and the Bitmap utility will

always try to let you use the larger size dot. Hence, if you make the size of the bitmap small, the frame may enlarge to let you use the larger dot, while if you make the bitmap large, the frame will adjust because you will have to draw using the small size dot. The size of the dot applies only to the magnified view of the bitmap. The dot size in the actual bitmap does not vary. You may adjust the size of the bitmap frame at any time, even after you have drawn a picture. If you make the box too small to hold the picture, part of the bitmap will be cut off, but it is not lost. Simply make the bitmap frame bigger again and you will find that your bitmap is still intact.

Once you have established the size you want your bitmap to be, you can start drawing. The drawing tool becomes a paintbrush when it moves over the bitmap frame. The tool can be one of three colors. When it is blue, it passes over the bitmap without drawing or erasing. To begin drawing, place the paintbrush over any empty pixel and press the left mouse button. The paintbrush will turn black, and anywhere you move it in the bitmap frame box, it will draw dots. To stop drawing, click the left mouse button again. To erase some of the dots, move the paintbrush on top of a previously drawn dot and press the left mouse button. The paintbrush will turn red, and anywhere you move it, it will erase dots. To stop erasing, click the left mouse button again.

There are two submenus at the top of the Bitmap Utility screen. The first is the GEOS submenu. Next is the options submenu. The first item under the options submenu is paste photo scrap. A photo scrap can be created and saved using geoPAINT. If you select this option, a box will appear showing all photo scrap files on the current disk. Click on one with the mouse, then click on OK to load the photo scrap into the bitmap, or click on Cancel to change your mind and not load a photo scrap. The next item is erase bitmap. This clears everything you have drawn from the bitmap frame. Be careful, this function does not verify that you really mean to clear!

USING BITMAPS IN YOUR PROGRAM

The most straightforward way to use a bitmap you have created in a program of your own is to display it on the screen. To do so, use the command:

10 BITMAP <STRING>,<XPOS>,<YPOS>

where the <string> is the name you gave your bitmap, <xpos> is the X position to place the bitmap on the screen and <ypos> is the Y position to place the bitmap on the screen. The <xpos> parameter can vary from 0 to 39, while the <ypos> can range from 0 to 199.

See below also for details of how to use bitmaps with Dialog boxes and Icon Lists.

DIALOG BOX EDITOR

Dialog boxes are used to present information and get a choice from the user. When geoBASIC is commanded to place a dialog box on the screen, it draws the dialog box in the center of the screen and waits for the user to click the mouse pointer in one of the buttons or on one of the icons in the dialog box. The box is then removed from the screen, and whatever was underneath it is restored. Finally, a number is returned indicating which choice the user clicked on.

STARTING THE DIALOG BOX EDITOR

To start using the Dialog Box editor from geoBASIC move the mouse pointer up to the Utilities submenu and click the left button. Move the mouse pointer to the submenu item dialog and click the left button again. The first screen of the Dialog Box editor will appear. In the box in the center will be listed any Dialog Box files which exist on the current disk in the drive. You may select one of these file to edit by clicking on it or type in the name of a new file under the prompt Create or edit an item Name:. Dialog Box names may be a maximum of 5 letters long. To continue, press

[RETURN] or click on OK. To return to geoBASIC just click on CANCEL.

USING THE DIALOG BOX EDITOR

The Dialog Box editor screen is split into two parts. On the left side of the screen, you can set the number and type of objects which will appear in the dialog box. On the right side, you set the placement and details of each item.

At the upper left side of the screen is a box with a number in it with the word Objects: next to it. This is the total number of objects in the dialog box. This number can vary from 1 object to 8 objects. Clicking in the up arrow on the left side of the number increases the total number of objects in the dialog box, and clicking in the down arrow on the right side of the number decreases the number of objects. As you increase and decrease the number of objects, the number of boxes which appear directly below the object number box will change. These boxes are used to set the type for each object which will appear in the dialog box. You may adjust the number of objects at any time.

Each object which appears in the dialog box can be one of a number of types, as detailed below. To adjust the type of a particular object, move the mouse to the object line and click on the up or down arrow to cycle through the different object types available. The various types of objects look different and some return numbers to the DIALOG call in your program, enabling the program to tell which object was selected to exit the dialog box. For each type of object, certain additional information needs to be specified on the left side of the screen. The appropriate prompts appear on the left side of the screen for each object line as you click on it with the mouse pointer. The object types are:

1. OK ICON — A button which contains the word OK. This is normally for the user to approve the last action taken by your program. The OK Icon returns the value 1 in the geoBASIC DIALOG function. Values which must be set on the right side of

the screen are the X and Y location of the icon.

2. CANCEL ICON — A button which contains the word CANCEL. This is normally for the user to cancel the last change made to the program and revert to the previous conditions. The CANCEL Icon returns the value 2 in the geoBASIC DIALOG function. Values which must be set on the right side of the screen are the X and Y location of the icon.

3. YES ICON — A button which contains the word YES. The YES Icon returns the value 3 in the geoBASIC DIALOG function. Values which must be set on the right side of the screen are the X and Y location of the icon.

4. NO ICON — A button which contains the word NO. The NO Icon returns the value 4 in the geoBASIC DIALOG function. Values which must be set on the right side of the screen are the X and Y location of the icon.

5. OPEN — A button which contains the word OPEN. This is normally for the user to open a disk file. The OPEN Icon returns the value 5 in the geoBASIC DIALOG function. Values which must be set on the right side of the screen are the X and Y location of the icon.

6. DISK — A button which contains the word DISK. Normally for the user to select the disk from various devices (such as the screen, printer, etc.). The DISK Icon returns the value 6 in the geoBASIC DIALOG function. Values which must be set on the right side of the screen are the X and Y location of the icon.

7. FIXED TEXT — This is a string constant which will appear in the dialog box. It is normally used to pass information to the user. Clicking on a fixed text string in the dialog box does not exit the dialog box and returns no value. Values which need to be set on the right side of the screen are the X and Y location of the text string and the text itself (30 characters maximum), which is entered on the Text: line.

8. VARIABLE TEXT — This is an expression which will appear in the dialog box. The expression can contain string and numeric constants as well as string variables and numeric variables. During the program, the string variables and numeric variables will be evaluated and their values used to put the variable text in the dialog box. Thus, the Variable Text field in a dialog box may be different each time the dialog box appears on the screen, unlike any of the other objects in the dialog box. Clicking on a Variable Text field in the dialog box does not exit the dialog box and returns no value. Values which need to be set on the right side of the screen are the X and Y location of the expression and the expression itself (30 characters maximum), which is entered on the String expression to use: line.

9. USER ICON — This is an icon designed using the Bitmap editor (see above). You must specify the X and Y location of the bitmap, the 5 character name that you gave the bitmap when you designed it, and the value returned to the DIALOG function if the user clicks on the bitmap. This value can vary from 20 to 255 and is entered at the Value to return: prompt.

SETTING OBJECT SPECIFICATIONS

As mentioned earlier, various information about each object must be specified on the right side of the screen. Both the Fixed Text and Variable Text objects require text lines, which are typed into the space provided for this purpose. The User Icon requires the value it will return to the DIALOG function. To adjust the number which appears in the Value to return box, click on the up or down arrow alongside the box. The number will "roll over" if you exceed the maximum (255) or try to go below the minimum allowed (20). The User Icon also requires the Bitmap name, which is typed into the space provided.

The X and Y positions must be specified for each object, and are measured as offsets from the upper left corner of the dialog box. The X position is changed by clicking in the up or down

arrows alongside the X offset: box, and the Y position is changed by clicking in the up or down arrows alongside the Y offset box. By default, the X and Y offsets will cycle through the default GEOS positions, which makes it easier to position the text and icons correctly. If you wish to have more freedom (within the confines of the dialog box) to position your object, click on the guides off item in the options submenu. You can adjust the Y offset to be anything between 1 and 95, while the X offset can vary from 0 to 176 in steps of 16 pixels. To return to using the default GEOS settings, click on guides on in the options submenu. Be careful not to set the X and Y offsets to be identical for two objects, since they will be drawn on top of each other, and one will be hidden from view.

OTHER MENU ITEMS

There are two submenus at the top of the screen. The first is the GEOS submenu. The other one is the options submenu. The items in the options submenu are:

GUIDES OFF/ON

The X and Y offsets for Dialog box objects will cycle through the default GEOS positions when guides are on. These positions make it easier to ensure that the objects don't overlap. When guides are off, the Y offset can be varied from 1 to 95, while the X offset can vary from 0 to 176 in steps of 16 pixels.

DISPLAY DIALOG BOX

This shows the current dialog box on the screen, except that any user-defined bitmaps are simply shown as blocks of pixels the size of the bitmap. Click on any button or bitmap to exit the dialog box and return to the editor.

QUIT

This exits the editor and allows you to save your dialog box definition, then exits back to geoBASIC.

USING DIALOG BOXES IN YOUR PROGRAMS

To display a dialog box designed with the Dialog Box editor on the screen in one of your geoBASIC programs, use the command DIALOG:

10 DIALOG STRING, VARIABLE

The string specifies the name of the dialog box as it was saved in the Dialog box editor. The variable, which is optional, will contain the number of the icon which the user clicked on when exiting the dialog box.

ICON LIST UTILITY

An Icon List is a group of bitmaps (maximum of 31) which can appear on the screen in your geoBASIC program. The bitmaps are designed using the Bitmap editor (see above). Once the bitmaps are on the screen, your geoBASIC program can branch to a specified subroutine if a bitmap icon is clicked on by the user. This ability to provide pictures on the screen from which the user makes a choice can enable you to write very easy-to-use programs.

STARTING THE ICON LIST UTILITY

To start using the Dialog Box editor from geoBASIC move the mouse pointer up to the Utilities submenu and click the left button. Move the mouse pointer to the submenu item icon and click the left button again. The first screen of the Icon List editor will appear. In the box in the center will be listed any Icon List files which exist on the current disk in the drive. You may select one of these files to edit by clicking on it or type in the name of a new file under the prompt Create or edit an item Name:. Dialog Box names may be a maximum of 5 letters long. To continue, press [RETURN] or click on OK. To return to geoBASIC just click on CANCEL.

USING THE ICON LIST EDITOR

At the top of the Icon List Editor screen is the Number of icons: box containing the number of icons that will be in the list. To increase the number of icons, click on the up arrow to the left of the number, and to decrease the number of icons, click on the down arrow to the right of the number. You can go back and readjust the number of icons in the list at any time.

Below the Number of icons: box is the Icon number box. This indicates the icon you are currently working on, and you can adjust this number from 1 to the maximum number of icons (as set above) by using the up and down arrows alongside the number. The X and Y positions, name of the icon to display, and the subroutine to branch to if the icon is selected refer to the icon number specified here. Next is the X position and Y position boxes. The X position can be set from 0 to 312 in increments of 8 pixels, while the Y position can be set from 0 to 199. Below the X and Y position boxes is a line for you to enter the name of the bitmap. This name is the one you gave it when you designed it using the Bitmap editor. Finally, Place to go: is for you to indicate which line number or label the program should branch to when the icon is clicked on. This location must be a subroutine. If nothing is specified here, then your program will not respond when you click on that icon.

There are two submenus at the top of the screen. The first is the GEOS submenu. The other one is the options submenu. The item in the options submenu is:

QUIT

This exits the editor and will allow you to save the changes, then exits back to geoBASIC.

USING THE ICON LIST IN YOUR OWN PROGRAMS

To use the Icon List created in your own programs is very simple: use the Icon command:

where "NAME" is the name you gave the icon list when you designed it using the Icon List Editor. The icons (bitmaps) will be placed on the screen in the chosen positions when this command is executed in a geoBASIC program.

SPRITE EDITOR

Sprites are graphic shapes which can be placed on the screen. The special property of a sprite is that it does not affect the graphics and text it passes over as it moves across the screen. Sprites are separate and independent of the screen graphics, which means you can create programs with sprites without having to worry about redrawing your background.

You may use up to six sprites in your program. Each sprite can be drawn in up to three colors and can be animated with multiple views (called frames). A starting position, X and Y velocity and the ability to link several sprites together into a larger shape can all be specified. Sprites can also be expanded (doubled in size) in the X direction, Y direction, or both. A position on the screen can be set, with the sprite moving from its initial position to this final position (called a "trail"). Finally, you may specify that after a given length of time, the sprite's motion be stopped and that a BASIC subroutine be called automatically. This is called a "timeout."

STARTING THE SPRITE EDITOR

To start using the Sprite editor from geoBASIC move the mouse pointer up to the Utilities submenu and click the left button. Move the mouse pointer to the submenu item sprite and click the left button again. The first screen of the Sprite editor will appear. In the box in the center will be listed any Sprite files which exist on the current disk in the drive. You may select one of these files to edit by clicking on it or type in the name of a new file under the prompt Create or edit an item Name:. Sprite names may be a

maximum of 5 letters long. To continue, press [RETURN] or click on OK. To return to geoBASIC just click on CANCEL.

If you choose the name of an existing sprite file, the next screen will request whether you want to Edit the sprite, Copy the sprite into a new sprite, Delete the sprite, or Cancel using the Sprite editor. Click on the item you want. The Copy option allows you to copy all the information about the sprite into a new sprite, so that you can make changes without affecting the original. If you select Copy then a new dialog box will appear for you to type in the name of the new file you want to copy the sprite data to. Type in the name and press [RETURN]. If you change your mind, click on Cancel .

USING THE SPRITE EDITOR

Once you have specified the name of the sprite file (either new or existing), the initial or Attribute screen of the Sprite Editor will appear on the screen. If the sprite is a new sprite, the minimum amount of information necessary for sprite definition will be placed on the screen. On the top line of the screen is the Sprite number. You may choose any sprite from 3 to 8 (sprites 1 and 2 are used by the system). To increase the sprite number, click on the up arrow to the left of the sprite number, and to decrease the sprite number, click on the down arrow to the right of the number. The sprite number will roll-over at sprite number 3 (to sprite number 8) and at sprite number 8 (to sprite number 3).

Just below the sprite number is some basic information about the sprite. First is the sprite Color. Although the sprites may be in up to three colors, two of those colors are the same for every sprite. Only one color can be varied, but this color may be different for every sprite. To cycle through the 16 available colors (black, white, red, cyan, purple, green, blue, yellow, orange, brown, light red, dark grey, grey, light green, light blue, and light grey) click on the up or down arrows alongside the sprite color.

Next to the Color box are the Double in X and in Y flags.

Clicking in the boxes will make the sprite twice as large in the X or Y direction. You may click on both boxes to double the sprite's size in both directions. Each sprite is 12 pixels wide by 21 pixels high. Realize that doubling the size of the sprite doesn't give you more pixels (higher resolution), it just makes the pixels you have larger.

The Start POSITION of the sprite is the final item on the Attribute screen. The starting X and Y coordinates can be set. To increase the X or Y coordinate, click on the up arrow to the left of the coordinate, and to decrease the coordinate, click in the down arrow to the right of the coordinate. The X coordinate can range from 0 to 380, while the Y coordinate can be from 0 to 256. The upper left corner of the screen is coordinate 31,31, so that sprites can move onto the screen from an off-screen position.

The rest of the items in the Sprite editor are available by selecting them from the two menus (in addition to the GEOS menu). When a submenu item has been specified for a sprite, an asterisk will appear next to the submenu item. Many of the options can be used together, although some options will deactivate others. These will be noted below. The submenu items under the New menu are:

SPRITE

This item is asterisked automatically because you are working on a sprite.

VELOCITY

The sprite velocity in the X and Y directions may be set. Velocity varies in increments of 8 pixels/second. To increase the velocity, click on the up arrow to the left of the X and Y. To decrease the velocity, click on the down arrow to the right. The X and Y velocities can range from –101 to 99, with negative velocities being motion to the left (X) or up (Y), and positive velocities being motion to the right (X) or down (Y). Setting a velocity of 0 for both X and Y causes the sprite to remain stationary. Velocity cannot be used with Trail.

PICTURE

For a sprite to be meaningful, it must have a picture. Otherwise, there would be nothing to see when the sprite was put on the screen. Clicking on this item adds a button marked Edit to the Attribute screen. Clicking on this button takes you to the Editor screen, where you can design your sprites, including multiple frame animations. The Editor screen is described below.

SEQUENCE

The ability to animate sprites is one of the most powerful features of geoBASIC . The animation is created by cycling through a series of frames to give the illusion of motion. Much as it is done in the movies, each frame shows a view of the object which is slightly different from the frame before. Each frame can be designed using the Editor screen.

Selecting this submenu item adds the animation information to the Attribute screen. The first item is how many ANIMATION Frames you want to use. You may have up to a maximum of 8 frames for each sprite. To increase the number of frames, click on the up arrow to the left of the number of frames. To decrease the number of frames, click on the down arrow to the right of the number of frames. The number you set here will determine how many frames are available for you to edit on the Editor screen. The ANIMATION Rate sets how fast the frames will cycle through the sequence. The rate measures how many 60ths of a second each frame will remain on the screen, and may vary from 1 to 240 (4 seconds). To increase the rate, click on the up arrow to the left of the rate. To decrease the rate, click on the down arrow to the right of the rate. The last item in this section is the Continuous ANIMATION button. If this is selected, then the animation will cycle continuously (like a man running). If this item is not selected, then the animation will cycle once and stop.

TIMEOUT

This item lets you specify the time interval between activating a sprite and calling a geoBASIC subroutine. You do not have to

use the timeout feature. The TIMEOUT Time sets the time delay before the BASIC subroutine is called. This is measured in 60ths of a second and can vary from 1 to 240 (4 seconds). The Stop motion at TIMEOUT flag sets whether the animation and motion will stop when the time runs out. If this item is selected, then the sprite will freeze in place at timeout, otherwise, it will continue to follow the instructions you set in the sprite editor file. To specify where the geoBASIC program will branch to when the time runs out use the At TIMEOUT gosub item. The line number or label of the subroutine is required, and the subroutine specified must end with a RETURN. When the subroutine has been performed, program execution will continue with the statement after the last one that was executed when the timeout occurred. Timeout cannot be used with Trail.

TRAIL

This submenu item lets you choose a position on the screen that the sprite will move to, as well as the velocity of the sprite. The program then takes care of all the complex calculations of figuring out the X and Y component of the velocity to get the sprite where you want it. If you had previously selected Velocity or Timeout, these selections will be replaced by the Trail data, since Trail cannot be used with those submenu items.

The X and Y coordinates of the position where the sprite is to end up can be set using the TRAIL to X coordinate and to Y coordinate. To increase the X or Y coordinate, click on the up arrow to the left of the coordinate, and to decrease the coordinate, click in the down arrow to the right of the coordinate. The X coordinate can range from 0 to 380, while the Y coordinate can be from 0 to 256. The upper left corner of the screen is coordinate 31,31, so that sprites can move onto the screen from an off-screen position. The Stop motion at trail end flag sets whether the animation and motion will stop when the sprite reaches the specified trail X and Y coordinates. If this item is selected, then the sprite will freeze in place when it reaches the trail point, otherwise, it will continue to cycle through its animation sequence. The At end gosub item specifies the subroutine that the geoBASIC program

will branch to when the sprite reaches the trail point. A line number or label is required, and the subroutine specified must end with a RETURN. When the subroutine has been performed, program execution will continue with the statement after the last one that was executed when the sprite reached its trail point. Unlike the other items in the New menu, Trail is a toggle. Select it once to turn it on, select it again to deactivate it and allow setting of Velocity and Timeout.

THE EDITOR SCREEN

The Editor screen is where sprites are actually designed. Each frame of the sprite must be drawn in order for it to be used in your program. To get to the editor screen, click on the Edit button in the lower right corner of the Attribute screen. If the button is not visible, select the item picture under the new submenu. You may also reach the editor screen by selecting the editor screen item under the options submenu.

The editor screen consists of two main portions. On the left side is the enlarged area for designing your sprite. The current sprite (as specified on the Attribute screen, above) will appear in an enlarged version when you enter the editor screen. Except when using linked sprites you must return to the Attribute screen to change the sprite you are working on. Whenever the mouse cursor enters the area on the left and is ready to start drawing, it will turn from its normal arrow shape to a paintbrush. To draw in the current color, move the mouse pointer over an empty pixel and click the left button. The mouse pointer will appear darker, and as you move the mouse pointer, it will fill in the pixels it passes over. To stop drawing, click the left button again. The pointer will appear lighter. To erase, move the mouse pointer over a pixel which is filled in and click the left button. The mouse pointer will appear darker, and as you move the mouse pointer, pixels will be erased.

On the right side of the Editor screen are various tools and information for your use. In the lower left corner are the three available colors for drawing the sprites. The color on the left is the

one which is different for the current sprite, and can be set using the Attribute screen. To choose a color for drawing the sprite, move the mouse pointer to the color you want to use and click the left button. A frame will be drawn around the currently selected color.

Directly above the color bar is the Current FRAME. When animating a sprite, each animation consists of several frames, up to a maximum of eight. You set the total number of frames for each sprite on the Attribute screen. The enlarged version of the sprite on the left side of the screen corresponds to the Current FRAME. To increase the frame number you are currently working on, click on the up arrow to the left of the current frame number. To decrease the frame number of the sprite you are working on, click on the down arrow to the right of the current frame number.

In the lower right corner of the Edit screen is a button marked Attr. Clicking on this button returns you to the Attribute screen. You may also return to the Attribute screen by selecting attribute screen item under the options submenu.

In the top portion of the right side of the screen, your animation runs continuously, showing the effects of any changes you have made. The speed at which this animation is running reflects the Animation Rate set on the Attribute screen.

In the center of the right side of the edit screen is information related to linking sprites. geoBASIC has the capability of linking up to four sprites together to form a larger, more complex shape than can be achieved by use of a single sprite. When sprites are linked, all motion of the sprite is controlled by specifying velocity, animation rate, etc., for the main sprite. The linked sprites will automatically follow the main sprite (which is counted as one of the linked sprites). To link sprites, click on the link sprites item under the options submenu. When you select this item, several choices will appear in another menu to the right of the link sprites item. You may select one of these items to link 2, 3 or 4 sprites. Whichever sprite you are currently editing will become the main

sprite which controls group movement. The sprites which will be linked are the next sprites in the numbering sequence. Thus, if you are editing sprite number 3 and want to link 4 sprites, then the sprites 3, 4, 5 and 6 will be linked. The current sprite must have a sprite number which is low enough that sprite number eight is not exceeded by any sprite in the sequence. You may not, for example, link four sprites beginning with number 6, since this would try to link sprite numbers 6 ,7, 8 and 9. There is no sprite number 9, so an error message will appear if you attempt this.

On the editor screen, the information for linked sprites appears in the center of the right portion of the screen. At the top is the Current SPRITE. Click on the up or down arrow alongside the current sprite number to cycle through the linked sprites. The current sprite is shown in the enlarged left portion of the editor screen, and its shape can be edited as described above. The Current FRAME item refers to the current frame of the current sprite, so by using both these controls, you may access any frame of any of the linked sprites.

When sprites are first linked, they are positioned on top of one another. If you wish to move the linked sprites to change their position relative to the main sprite, use the X Off or Y Off options under the LINK Offsets heading. Use the Current SPRITE option to set the sprite whose offset you want to adjust (you cannot adjust the offset for the main sprite). Then click on the up arrow to the left of the X Off to increase the X offset of the sprite (move it to the right of the main sprite) or click on the down arrow to the right of X Off to decrease the X offset of the sprite (move it to the left). The sprite can be positioned to the left of the main sprite by setting the X Off to a negative number. To adjust the Y offset of the sprite, click on the up arrow to the left of the Y Off to increase the Y offset of the sprite (move it below the main sprite) or click on the down arrow to the right of Y Off to decrease the Y offset of the sprite (move it up). The sprite can be positioned above the main sprite by setting the Y Off to a negative number.

In addition to the attribute screen and link sprites items in the

options submenu (discussed above), the following are available:

PASTE FRAME

This item will paste the contents of another frame of the current sprite into the current frame. This can be very handy when the frames of your animation require only small changes from one frame to another. Selecting this menu item brings up a dialog box requesting the number of the frame whose contents you want to paste into the current frame. Enter the number of the frame and press [RETURN] or click on the Cancel button to abort. If you select a frame number which does not exist, then an error message will appear to let you know that.

QUIT

This menu item exits the editor and will allow you to save all your changes, then returns you to geoBASIC.

Chapter 8—The geoBASIC Debugger

The geoBASIC debugger allows you to check the values of variables, stop the program on certain lines or when variables reach certain values, and single-step through your code. For those who are used to debugging by printing out variables, the debugger will take a little getting used to, but you will find it very powerful with practice. The debugger is loaded with the geoBASIC program, so programs may run slightly slower when you are in debugging mode.

To debug a program from the editor, click on the DEBUG menu option. The debugger dialog box has four fields for displaying the values of variables or breakpoints in the program.

There is a menu in the debugger box for setting the current debugging mode and choosing what to display in the field boxes. Clicking OK will start the program running. Clicking CANCEL will take you back to the editor. To return to the debug box while a program is running, hit the RUN/STOP key.

ENTERING EXPRESSIONS OR BREAKPOINTS

Clicking in any of the four field boxes will allow you to enter or edit an expression or a breakpoint. By default, expressions are displayed whenever the debugger dialog box comes up. To enter an expression, click on one of the boxes. Enter the name of a variable. When you press return or click on another box, the value of that variable will be displayed to the right of the box. This is similar to typing PRINT in a Commodore program after a program has been running. Normal arithmetic operators can be used, +, -, etc., but functions may not. Logical operators such as AND, OR or NOT can be used, but you must use the single characters &, |, and ! respectively (| is C= ^). This is a little bit confusing but useful in debugging.

Some examples of legal debugger expressions and their

BASIC equivalents:

X	**X**
X + Y	**X + Y**
(2 * X) + Y	**(2 * X) + 7**
(X<10) & (X>15)	**(X<10) AND (X>15)**
(X<10) | (X>15)	**(X<10) OR (X>15)**
X <> 5	**X <> 5**
! (X <> 5)	**NOT (X <> 5)**
X(5)	**X(5)**
3.14 * X	**3.14 * X**
X ^ 3	**X ^ 3**
A$	**A$**
A$ + B$	**A$ + B$**

Some illegal expressions:

SIN(X)
SQR(X)
LEFT$(A$)
ABS(X+Y)
LEN(A$)

To change something in a field box, click on it and edit it. When you press return the changes will be reflected in the result. Expressions can be up to 15 characters in a given field.

The debugger supports two kinds of breakpoints, line breakpoints (ln brk) and expression breakpoints (ex brk). Breakpoints are places in the program where you would like to stop execution and look at the values of variables. Hitting cl brk on the submenu of "show" will clear the breakpoints.

Line breakpoints allow you to specify up to four line numbers that the program will stop at after it executes a line. Type in the following program and select DEBUG:

```
10  A = 3
20  B = 2
30  B = 4
```

The debugger box will come up immediately, with four blank boxes for displaying expression values. To set a line breakpoint, choose "line break" in the display menu. Now the four fields will display and allow editing of four line breakpoints. Type "20" into one of the boxes. Click OK. The program will execute lines 10 and 20 and return to the debug box. You can use the display boxes to print out the values of a and b, if desired. If you wish to set another line breakpoint, choose "line break" again and enter another line number to break at. If you do not delete the old breakpoint, it will still be active. NOTE: Line breakpoints break at the completion of the entire line and effectively ignore RETURN statements. If you are still displaying line breakpoints and want to display expressions again, choose "values" in the display menu. When the program ENDs, you will be returned to the editor again, as if running a normal program.

Another kind of breakpoint is an expression breakpoint. This allows you to specify up to four expressions to break on. If any of the expressions are true after executing a line of the program, the program will halt execution and return to the debug box. Start execution of the above program by clicking on OK. When the debug box comes up, choose "expr break" in the display menu. Type "b > 2" in one of the field boxes. When you press OK again, the program will halt when the variable b exceeds 2, which should happen after executing line 30. This is especially useful when you have an "out-of-range" error for an array subscript. You can set up an expression breakpoint to stop the program when your subscript goes over the maximum value it can be. The line you stop on should be the line that causes the problem.

If an expression breakpoint occurs, an asterisk "*" is placed to the left of the first expression that caused it (there may be more than one true expression).

You can clear a particular breakpoint simply by editing it and deleting everything or you can clear all the breakpoints by selecting CL BRK in the SHOW submenu. This is useful if you used the RUN or TRACE mode, a breakpoint was encountered and

you would like the program to continue without stopping any more.

DEBUG MODES

There are three debugging modes, RUN, STEP and TRACE. RUN allows the program to execute normally and the debugger dialog box only appears if a breakpoint was found, so it is like RUNning a Commodore BASIC program with STOP commands imbedded in it. The default debug mode is RUN, and clicking OK starts up the current mode. If you hit BREAK while the program is running, the debugger dialog box comes up and you can set more breakpoints or change the expressions being displayed in the windows.

STEP allows you to single-step through your program command by command. The debug box will reappear after each command is executed. It redisplays your expressions (if any) as the program progresses. For example, if you had a FOR-NEXT loop, you could put the name of the index variable in a display field and watch it increment as the loop is being executed. To continue stepping through your program, click on OK or "step" on the mode menu. The line number at the bottom is updated after each command has been executed—therefore, if you have multiple commands on one line, the line number will not change until the next line is encountered.

The TRACE option is like RUN, but if the program encounters a breakpoint the debug box is displayed only briefly, then continues execution. Hence you can display the value of a variable in a loop without hitting OK each time. To get out of TRACE mode, hit the break key. The debug dialog box will return and will automatically be set to STEP mode.

ERRORS

If, while running the debugger, an error occurs in your program, a dialog box with "Check #" and the error number will

come up. Clicking OK will bring up the debugger dialog box and you can display variables to see what went wrong. If you hit OK at this point, the debugger will attempt to continue if the error was not fatal.

Chapter 9—geoBASIC Error Messages

The following is a list of error messages geoBASIC can generate. They are listed in numeric order.

Number	Error message	Description
0	Syntax	a typo, number of parameters wrong, etc.,
1	Out of blocks	see the GEOS Programmer's Reference Guide
2	Bad track	see the GEOS Programmer's Reference Guide
3	Disk full	there is no more room on the disk to SAVE or CREATE, etc., a file
4	Full directory	see the GEOS Programmer's Reference Guide
5	File not found	OPENing, LOADing, etc., a nonexistent file
6	Bad BAM	see the GEOS Programmer's Reference Guide
7	Unopened file	a file must be OPENed before APPEND, DELETE, etc., can be done on it
8	Bad Record	this record does not exist yet so use an APPEND or an INSERT at the correct record number
9	Out of records	the limit on a VLIR file is 128 records so no more INSERT or APPENDs can be done
10	Bad structure	see the GEOS Programmer's Reference Guide
11	Buffer overflow	generally happens if a file is read past its end, i.e., with RDBYTE, etc.,
12	No print driver	a printer driver was not selected or not on the current disk drive with geoBASIC
13	Device not found	the printer is either off-line or turned off
14	Next without For	a NEXT statement was found before a FOR command
15	Return without Gosub	a RETURN statement was found before a GOSUB
16	Out of data	there was not enough data in DATA statements to keep up with all the READ commands
17	Illegal quantity	an illegal value was used for something, i.e., using 320 as a parameter for the LINE command but the right

		maximum is 319, etc.,
18	Overflow	the result of an expression was too large or too small for geoBASIC to handle
19	Out of memory	there was not enough memory to allocate a new array, string or load in a new font
20	Undefined Statement	the line number for a GOTO, GOSUB, etc., does not exist
21	Bad subscript	the value given an array subscript is negative or larger than in a DIM statement
22	Redimensioned array	an array with the same name was already defined
23	Division by zero	an attempt was made to divide by zero, most likely an uninitialized variable
24	Type mismatch	an attempt was made to assign a string to a floating point variable or vice-versa without using VAL or STR$
25	String too long	string expressions must have 0 to 255—if the resulting string is longer (i.e., adding two 200 byte strings together) then this error ensues
26	Formula too complex	formula too hard to evaluate—most likely when a string expression has three or more sub-expressions
28	Undefined function	a function was used without first using DEF FN
30	Redefined label	a label already exists with the same name—check the first six characters again
31	Line too long	a line was greater than 240 characters
32	No header	see the GEOS Programmer's Reference Guide
33	No disk	the disk is not in the drive
34	No data block	see the GEOS Programmer's Reference Guide
35	Bad data	see the GEOS Programmer's Reference Guide
37	Verify	see the GEOS Programmer's Reference Guide
38	Write protect	an attempt was made to write to the disk with the write protect notch covered
39	Bad header	see the GEOS Programmer's Reference Guide
40	Font too big	the font was too big to fit in memory
41	Wrong disk	the disk that was in the drive was accidentally taken out and replaced with another one
42	Label not found	the label used as an argument was not defined or the object data does not exist

43	Labels full	there can only be 127 labels
44	Loop not found	a LOOP or UNTIL was not found after a REPEAT or WHILE command was executed
45	Loop without do	a LOOP or UNTIL was found before a REPEAT or WHILE command was executed
46	Byte decode	see the GEOS Programmer's Reference Guide
47	Break	RUN/STOP was pressed to halt the program
48	Too many processes	there can only be 8 processes running at a time to start up a new one, an old one must be deleted first
49	DOS mismatch	see the GEOS Programmer's Reference Guide

NOTES

NEED HELP?

Your best source of technical assistance is on the QuantumLink online network. *RUN* maintains a geoBASIC message board on Q-Link to provide users with timely answers to questions and comments. Check it out even if you don't need assistance. The *RUN* geoBASIC area also provides users with programming tips and information, sample applications and programs to download.

If any manufacturing defect becomes apparent, *RUN* will replace the defective disk free of charge if returned by prepaid mail within 30 days of purchase. Send it, with a letter specifying the defect, to

RUN Special Products
80 Elm Street
Peterborough, NH 03458

Replacements will not be made if the disk has been altered, repaired or misused through negligence, or if it shows signs of excessive wear or is damaged by equipment.

DON'T MISS THESE OTHER SPECIAL DISKS AND PRODUCTS AVAILABLE FROM *RUN*:

GEOS COMPANION—*RUN*'s GEOS "creativity" disk with music and animation programs, clip art and fonts included among the ten powerful programs on this disk. $24.97.

GEOS POWER PAK II—GEOS enhancements, accessories, applications and utilities, including the latest GEOS telecommunications program, text editor, games and more. For both C-64 and C-128 GEOS users. $24.97.

GEOS POWER PAK I—Features ten application and utility programs, over 20 fonts and over 100 clip art images. For C-64 GEOS. $24.97.

SUPER STARTER PAK—Seven powerful applications to meet all your computing needs—word processing, database, spreadsheet, terminal program, paint and draw, DOS shell, mailing label. For both C-64 and C-128 modes. $24.97.

FUN PAK 128—A special collection of eight games—including an exciting space adventure, arcade and role-playing games and brain-teasing challenges—that take advantage of the C-128 mode capabilities. $19.95.

PRODUCTIVITY PAK III—Everything C-64 and C-128 users need on one disk—including the famous RUN Script word processor. $19.97.

RUN WORKS—Seven of *RUN*'s most powerful programs, including RUN Paint, RUN Term, Graphmaker and Money Manager. For both C-64 and C-128. $24.97.

PRODUCTIVITY PAK I—Fifteen of *RUN*'s best applications and utilities for C-64 mode computerists. Includes the powerful database, DATAFILE, and accompanying mailing label and spreadsheet programs. $19.97.

VOICEMASTER JR.—Now your Commodore can speak your language with this speech output and voice recognition peripheral that lets you control programs or other peripherals with simple spoken commands. $19.95.

To order call *RUN* at **800-343-0728** or send check or money order to ***RUN* Special Products**, 80 Elm Street, Peterborough, NH 03458.