

GFA BASIC

Programmers Reference Guide, Volume I

An Advanced Guide to the
GFA BASIC Interpreter

George W. Miller



GFA BASIC Programmers Reference Guide, Vol. I™

A Guide to GFA BASIC

**For the Atari ST Series of Personal
Computers**

**Written by
George W. Miller**

**Published by MICHTRON, Inc.
576 South Telegraph
Pontiac, Mi. 48053**

**(313)334-5700
BBS: (313)332-5452**

GFA BASIC Programmers Reference Guide Vol. I™

Published in the U.S.A. by MICHTRON, Inc.

576 South Telegraph
Pontiac, Michigan 48053

(313) 334-5700
BBS: (313)332-5452

Written by George W. Miller

Cover Design by Thomas Logan

Copyright 1988, MICHTRON, Inc.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 and 108 of the United States Copyright Act without the permission of the copyright owner is unlawful.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

1988

ISBN 0-944500-08-0

The opinions expressed in this book are solely those of the author and are not necessarily those of MICHTRON, Inc.

Further Trademark and Copyright Notices:

MICHTRON is a registered trademark of MICHTRON, Inc. GFA and GFA BASIC are registered trademarks of GFA Systemtechnik. Atari, 520ST, 1040ST, Mega, NEOchrome and TOS are registered trademarks of ATARI Corp. Degas is a trademark of Batteries Included.

Disclaimer of Warranty and Limits of Liability

The author and publisher of this book have used their best efforts in preparing this material and the programs contained in this book. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness.

The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this book.

The author and publisher shall not be liable for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

Help!

Many people have difficulty typing listings correctly from published material. It's quite common for a user to type an incorrect number or letter, then not be able to identify the problem area when the program fails to function properly.

This problem is even more common with GFA BASIC listings. The absence of line numbers makes it even harder than normal to transfer a printed listing to your computer.

We suggest that if you encounter a problem with any program in this book, you follow several simple steps *before* you call us. First, check any data statements at the end of the program for a mistyped number. This is the most commonly encountered error.

Second, have another person read you the listing, line by line, from the book, while you check each program line on your monitor.

In Chapters 3, 4, 5 and 6, where we expect you'll want to type in some of the program examples, we've placed a special symbol at the point in each line where you should press the <Return> key.

This is necessary because sometimes the length of the actual line in the GFA BASIC Editor is longer than can be printed on one line in a book. Watch for the <Return> symbol, ↵, which will be placed at the end of each program line. A line will look like this:

```
PRINT "This is a sample line." ↵
```

In Chapter 2, this symbol was omitted, and each line in the short example program listings is presented in a form similar to the way it will appear on your screen.

Also, the code in each of the major programs is broken up into modules, with a brief explanation. Be sure you type each program in completely before trying to run it.

If, after carefully comparing the lines you have entered to the listings in the book, you still haven't found the problem call or write us. We may have additional suggestions for you.

If you'd rather not type in the program listings, we have all of the major programs in this book available on disk for an additional fee. A special coupon is located at the end of the book.

Acknowledgements

Writing a book takes quite a commitment. Only those who have labored on projects such as this can truly appreciate the necessary sacrifices.

Not my sacrifices; I was really interested in sharing what I had discovered about GFA BASIC with others, but the sacrifices made by my family — the evenings and weekends when other activities had to be put aside as my deadlines approached.

I'd also like to thank my wife, Carol, for putting up with stacks of reference material, note paper, and the general total disorganization of my computer room for months on end. Maybe now I can organize it, before my next project takes over.

I appreciate the patience of my sons, who managed to understand that I had work to do, and often couldn't spend time playing ball with them, and my daughter, who managed to keep the TV turned down to a reasonable level.

Further appreciation is extended to Gordon Monnier and Laine Reynolds, the President and Vice President of MICHTRON, Inc., who had confidence in me and offered encouragement when times got rough.

Finally this book is dedicated to the memory of my father. No matter what I tried to do, he always believed I could succeed. He would have been proud to see this book.

In memory of my father,

George Warren Miller
October 21, 1923 - June 11, 1986

From the Author

This book was intended to be the ultimate source for information about GFA BASIC. Everything anyone could ever want to know was to be included.

It didn't take long before I discovered that such a book just wasn't possible. The book would be several times the size of this one. Therefore, this is the first part of an opened ended set of reference books for GFA BASIC.

Topics I wanted to cover in this book, information which would have required an additional 300-400 pages, and increased the cost of the book considerably, will be covered in later volumes.

Also, future releases of GFA BASIC will be explained as needed. No information will be repeated in later volumes, except as absolutely necessary. An effort was made to ensure that this volume would not be outdated by future releases of GFA BASIC.

The GFA Programmers Reference Guide, Vol. I contains something for everyone; whether you are a novice programmer, or an experienced professional.

Thank you for your interest in GFA BASIC, and the GFA Programmers Reference Guide, Vol. I.

George W. Miller

About the Author:

George W. Miller was formerly an Assistant Editor and Programmer for COMPUTE! Publications, and was one of the first people to recognize the power of GFA BASIC.

Mr. Miller is presently the Director of Product Support for MICHTRON, Inc.

Table of Contents

Table of Contents

Foreword	1
Programming In GFA BASIC	4
The Same, But Different	7
Structured Programming	7
A Closer Look	8
GFA BASIC Variables	8
Mathematical Functions	10
Hierarchical File Structures	11
 Chapter 1	 13
The GFA Editor/Interpreter	13
The Command Line	16
Save	16
Load	16
Save,A	16
Merge	17
Quit	17
New	18
Block	18
Blk Sta	19
Replace	19
Find	20
Pg up	20
Pg down	20
Text 16	20
Text 8	20
Insert	21
Overwrt	21
Direct	21
Flip	21
Run	21
Test	21
Keyboard Commands	22
 Chapter 2	 25
The GFA BASIC Language	25

Chapter 3	263
GFA Graphics	263
Graphics Capabilities	265
Graphics Programming Techniques	265
Resolving a problem	269
GFA BASIC Graphics Commands	270
String Art	273
Kaleidoscope	285
Program 3-3. KALEDIO.BAS	285
The Dragon Plot	293
The Mandelbrot Set	296
Saving a Screen in NEOchrome Format	310
Great Graphics	317
Chapter 4	320
Simple Animation	320
The Animation Sequence	320
Screen Flicker	321
Page Flipping	321
First Steps	321
Bouncing Balls	321
Orbit	326
Shape Editor	328
Using the Shape Editor	348
Changing the Palette	351
Using The Code	352
A Minor Detail	352
Chapter 5	356
Sound	356
The Physics of Sound	356
The Sine Wave	357
Understanding Waveforms	357
The Sound Command	358
The Sound Envelope	360
WAVE	361

Chapter 6	390
Input/Output	390
Telecommunications	390
Using EasyTerm	477
Status Display	477
A Guided Tour	478
EasyTerm Functions	478
EasyTerm DOS	480
Auto Dial	480
Printer Options	481
Set time and Date	481
Define Function Keys	482
Disk Write Verify	482
Help	482
Script Files	482
Xmodem - Getting Technical	484
Appendix A	A-1
Error Messages	A-1
GFA BASIC Error Message	A-2
TOS and GFA BASIC Compiler Error Messages	A-4
68000 Exception (Bomb) Error Messages	A-5
Appendix B	B-1
BIOS Functions	B-2
Appendix C	C-1
XBIOS Functions	C-1
Appendix D	D-1
GEMDOS Functions	D-1
Appendix E	E-1
GFA BASIC Editor/Interpreter Quick Reference Table	E-2
Index	I-1

Foreword

Introduction to Programming with GFA BASIC



Foreword: Introduction to Programming with GFA BASIC

In 1985, Atari Corporation introduced the consumer to a new and powerful computer, The Atari 520 ST. With it's MC68000 microprocessor, a 16 bit chip, an 8 Megahertz clock, 524,288 bytes of random access memory (soon to be expanded to one megabyte and beyond), and the GEM operating system from Digital Research, Inc., the Atari 520ST was hailed by many as a bold new vehicle, allowing the general computing public to explore a new world of computing power.

The ST was a grand departure from the traditional personal computer. Here was a personal computer offering previously unheard of power. And, it was also one of the first computers aimed at the mass market that was sold without a version of BASIC stored in ROM (Read Only Memory) chips.

The original Atari ST Developers Systems arrived with a C compiler, but no text editor. The acquisition of an editor was left up to the individual.

Additionally, the official documentation from Atari consisted of reams of photocopied pages, most of which applied to GEM on the IBM PC, not the 68000 version of GEM.

For most people, it was very difficult to make any progress in learning the secrets hidden within this strange, new machine.

The BASIC (Beginners All purpose Symbolic Interpreted Code) language was only a promise from Atari. If you wanted to program your ST, you had to learn a new language.

Many of the original ST programmers were skilled in machine language on the 6502 and 8088 microprocessors, and well versed in BASIC, so learning a new language was only a minor inconvenience.

Many developers attempted to fill the void by offering languages for the ST: versions of C, Pascal, Modula 2 and Forth appeared for the ST in rapid succession.

Still, many experts felt that in order for the ST to make a significant impact in the home computer market dominated by the hobbyist who dabbled at programming, a version of BASIC was necessary.

Early in 1986, Atari Inc. introduced ST BASIC, written for the ST by Metacomco, and a version of LOGO. This was the moment we had been waiting for. Many of us rushed to explore this new BASIC, eager to develop the power that a machine such as the ST was sure to unlock.

ST BASIC was a bitter disappointment. The environment was anything but friendly, and even worse, this BASIC had many bugs, some quite severe. LOGO was also disappointing. It was totally inadequate for all but the simplest programming tasks.

Once again, the only choice was to learn a new language.

Many programmers had limited resources, able to afford only one floppy disk drive. Since compiled languages required large libraries of routines to be accessible when compiling code into an executable program, this necessitated extensive disk swapping.

Lengthy compile times were also the rule. A program would be edited, compiled, linked, then finally run, only to discover a minor flaw in the program. Then it was necessary to load the editor program, correct the source code, and repeat the process. Often 15 minutes could be wasted between editing a single line of source code

and executing the object code. In fact, so much time was required that, with distractions, it was possible to actually forget what changes you had made in the code.

Additional RAM helped somewhat, giving the user the ability to store files in a RAM disk, but the process was still clumsy. Until hard disks arrived for the ST, the process would continue to frustrate novice programmers.

It was much more fun to program in the interactive environment of the old 8-bit computers, with their built-in BASIC's. And I began to doubt my wisdom in purchasing an ST for my next generation of computer.

Then, one day, late in 1986, after I had decided that I would program my ST in C and Assembly Language, I saw a program written by a programmer in Germany.

Incredibly, this program claimed to have been written in BASIC, but the source code was for a version of BASIC I had never seen before. The program wasn't memorable, but what it did, it did FAST!

This program was written in GFA BASIC. Further research indicated that GFA BASIC was owned by GFA Systemtechnik, located in Dusseldorf, West Germany. Soon I had what was probably the first copy of GFA BASIC in the United States.

The documentation was all in German, but GFA BASIC version 1.0 was a revelation of what the ST could do with an interactive language.

The commands were quite similar to the BASIC I was already familiar with. After a few weeks of experimenting, I was converted. I began extolling the virtues of GFA BASIC and wishing that GFA BASIC would be distributed in the U.S. so others could profit from this unique and powerful language.

I also had an ulterior motive. At the time I was employed by COMPUTE! Publications. COMPUTE! wanted to support the Atari ST with articles, but the editors didn't want to use C or assembly language articles. They felt that there weren't enough programmers using either to warrant including code in the magazine.

I was limited to writing only in ST BASIC for COMPUTE! magazine. Of course by this time I had decided that ST BASIC wasn't for me. In fact, I was refusing to have anything to do with it. I would often jokingly state that using ST BASIC had been proven to cause brain damage.

Since using ST BASIC was totally unacceptable to me, perhaps GFA BASIC would be the catalyst needed by the ST.

Fortunately, the developers of GFA BASIC felt there was a market for their product in the United States, and decided to try to find a U.S. distributor.

Atari is reported to have turned down GFA BASIC because they already had an admittedly inferior version of BASIC, and besides, GFA BASIC "wasn't really BASIC, because it didn't use line numbers."

Luckily for all of us, among the people GFA approached was Gordon Monnier, President of MICHTRON, Inc., who saw the value in GFA BASIC.

At last a simple, but powerful interactive language was available for the Atari ST!

GFA BASIC

Frank Ostrowski, the programmer responsible for GFA BASIC, has stated that after he saw the inferior BASIC shipped with the ST, it was possible to develop a new BASIC, which didn't have to conform to the standards of any other BASIC interpreter.

Frank's goal for this new language was to provide the simplicity of BASIC, and the ability to write structured code.

Structured programming is best defined as the construction of programs from smaller programs that are either structured programs themselves, or made up of a number of particularly well understood control structures.

As a bare minimum requirement, a structured language must contain an IF THEN ELSE decision branch, and at least one loop structure based on boolean logic, such as the REPEAT UNTIL, WHILE WEND, or DO LOOP structures which are all included with GFA BASIC.

The first step was to eliminate line numbers and find a solution to the problem of resolving GOTO's and GOSUB's. Frank also felt it was important to be able to pass parameters to subroutines (known as Procedures in GFA BASIC), and to declare local variables, permitting a programmer to use recursive code.

GFA BASIC is written entirely in machine language, making it a fast and compact language. Version 1.0 of GFA BASIC took only 6 months from conception until it was a reality.

Everything was done with an eye on eventually compiling the resulting code, creating an even faster program, executable from the GEM desktop.

Although bearing many similarities to languages such as C, Pascal, and Modula 2, GFA BASIC is in reality a hybrid version of BASIC. Programmers who attempt to attack a GFA BASIC project from the view point of a C, Pascal or Modula 2 programmer are doomed to frustration.

Remember that GFA BASIC is a unique version of BASIC. In fact, it would be better if this language was called simply GFA, with no reference to its heritage in BASIC.

If you follow the simple rules for GFA BASIC, you'll unleash the astonishing power at your finger tips in no time.

Programming In GFA BASIC

The purpose of this book is to introduce the new concepts of GFA BASIC and provide a through explanation of every command, suggest methods to be used in writing your own programs, and provide a needed reference book for the Atari ST version of GFA BASIC.

This book doesn't teach GFA BASIC. If you have no previous programming experience, use this book in conjunction with a good book on programming in BASIC. Take programming examples from other sources, and translate the source code to GFA BASIC, referring to this book for descriptions of the many commands available.

As your level of programming competence increases, you'll find other useful information at your finger tips.

This section introduces GFA BASIC, and explains a few simple rules for programming. The following chapters pave the way, starting you down the road to creating your own more powerful programming applications.

Some programmers using other high level languages look down their noses at programmers using the lowly BASIC language. With GFA BASIC, you'll be able to write any application (with the exception of Desk Accessories) that can be created with any other language. As an added bonus, you'll waste less time and expend less effort than other languages require.

- **Chapter 1, The GFA Editor/Interpreter** details how to get the most out of the Editor/Interpreter.
- **Chapter 2, Commands and Functions**, is an analysis, with example routines, of every word in the GFA BASIC language. This section alone is larger than the manual which was shipped with GFA BASIC.
- **Chapter 3, GFA Graphics**, introduces graphic concepts, and provides interesting programs, as well as including Procedures which may be utilized in your own programs.
- **Chapter 4, Computer Animation**, introduces Sprites, and the necessary principles for writing animation routines.
- **Chapter 5, Sound and Music**, illustrates uses for the built-in sound chip.
- **Chapter 6, Input/Output**, uses a telecommunications program to demonstrate sending and receiving data via the RS-232 and printer ports.

The Appendices provide perhaps the most important information in this book for a programmer. Here you'll find a treasure chest of invaluable information; complete information on BIOS, XBIOS, and GEMDOS functions, as well as other information. You'll find yourself referring to these sections constantly as you continue to gain skill as a GFA BASIC programmer.

Getting Down to (GFA) BASIC's

BASIC is the most widely used computer programming language today. It's available in some form for nearly every computer. Many computers even have a version of BASIC stored in ROM, making it by default, a standard language.

In spite of this, BASIC doesn't enjoy a very good reputation. Computer Science courses stress Pascal over BASIC. One very good reason for the preference of Pascal over BASIC is that Pascal requires the program to be written in a highly structured manner.

This makes it easy for an instructor to decide what's right and what's wrong, when actually there is no right or wrong. Only code that works and code that doesn't work. The program is either properly structured, or it's not.

The concept behind the development of GFA BASIC was not just to make a few minor improvements to existing versions of BASIC, but to develop a totally new version of BASIC.

Among the advantages of GFA BASIC are:

- GFA BASIC supports structured programming.
- It's easy for anyone familiar with another version of BASIC to begin using the interpreter.
- All the advantages of programming in traditional versions of BASIC are present in GFA BASIC.
- The GFA BASIC interpreter is compact, consisting of only about 55,000 bytes of code in the present implementation.
- GFA BASIC is fast, often able to achieve speeds which rival pure assembly language.
- The mathematical functions of GFA BASIC are accurate to 11 digits.
- Perhaps the most important advantage, the programmer is able to develop and test his creation in an interactive programming environment, then compile the final result into a stand alone program.

The Same, But Different.

Programming in GFA BASIC will require some adjustment from programmers familiar with other BASIC's.

The first difference you'll notice is that there are no line numbers. BASIC programmers are used to seeing line numbers. How can you use GOTO and GOSUB without line numbers?

GFA BASIC supports the use of labels for resolving GOTO and GOSUB statements. These labels make it possible to move to every section of the program, within certain natural limitations. One such limitation is that a Procedure (the proper name for a subroutine in GFA BASIC) may only be entered at the beginning and must be exited through the natural end of the routine.

Line numbers in traditional BASIC's also allow commands to be inserted into a program segment, sections of a program to be deleted, and the program to be listed selectively. All these functions of BASIC are accomplished through GFA BASIC's powerful editor. You will learn how to take advantage of the Editor's features in Chapter 1.

Line numbers in traditional BASIC's actually may create problems. When a program is modified, it may need to be renumbered. Also, the process of referencing line numbers is tedious, and error prone. Which of the following program lines is more understandable?

GOSUB 1010

or

GOSUB format_disk

By using meaningful names for labels and procedures, GFA BASIC is somewhat self-documenting, producing understandable code.

Also, by eliminating line numbers, GFA BASIC eliminates the necessity to use memory for storage of the numbers and line links. Simply, GFA BASIC makes more efficient use of the available storage space.

A further major difference from most traditional BASIC's is that GFA BASIC permits only one command on each program line.

Structured Programming

As previously mentioned, structured programming may be defined as the construction of programs from smaller programs that are either structured programs themselves, or made up of a number of particularly well understood control structures.

GFA BASIC implements structured programming by adding several structure commands to BASIC:

DO..LOOP
WHILE..WEND
REPEAT..UNTIL
GOSUB...PROCEDURE

These commands are explained in detail in Chapter 2.

A Closer Look

GFA BASIC improves on many of the features of existing versions of BASIC, and includes a few new features not found in *any* version. In the remainder of this chapter, we'll take a look at a few principles you should understand to get the most out of this book.

Throughout *The GFA BASIC Programmers Reference Guide, Vol. 1*, the many examples will be listed with the GFA BASIC keywords capitalized and variables and labels in lower case letters. This is an option for GFA BASIC listings, and is available by typing DEFLIST 0, as explained in Chapter 2.

The following program segment illustrates the general form of a GFA BASIC program.

```
PRINT "Pick a number between 0 and 9";
INPUT a$
IF VAL(a$)>9 OR VAL(a$)<0
  PRINT "Wrong choice!"
  errflag!=FALSE
ENDIF
IF errflag!=FALSE
  GOTO start
ENDIF
PRINT "You chose ";a$
```

GFA BASIC Variables

Many versions of BASIC limit the number of characters in a variable name to only a few. Even those versions of BASIC which permit longer names may only use the first few characters for identification. GFA BASIC permits up to 255 characters, or the length of one line in the editor, with all characters in the variable name being evaluated as part of the name.

All variable names must begin with a letter, but after the first letter, any combination of numbers, characters, the slash or underline character are permitted.

Some sample variable names would include:

```
variable_1  
variable_2  
name$  
first_name$  
last_name$  
another_rather_long_but_legal_variable_name
```

Using meaningful variable names simplifies programming. Instead of using $x=100$, a variable name such as, `class_members=100` is certainly clearer in its meaning.

One caveat to remember, be certain you spell the variable names the same way every time. You know that colour and color are the same thing, but to GFA BASIC, they will be interpreted as two different variables. This type of error is actually quite common, and may be very difficult to locate and correct.

Even keywords may be used as variable names. COLOR is a keyword in GFA BASIC, but LET color=1 is also legal definition for a variable.

By using the optional LET statement, all but a few reserved words may be used as variable names in GFA BASIC.

Variable Types

Four distinct types of variables are used in GFA BASIC, string, real, integer and boolean variables. Each variable type may be further defined into arrays of variables. (For more information on arrays, see the entry for the DIM statement in Chapter 2.)

Furthermore, a variable may be declared to have meaning throughout the program, or it may be defined as a local variable, having meaning only within a specific procedure in a program. The same variable name could have different meanings at different points within the same program.

Variable Types

<u>Variable</u>	<u>Type</u>	<u>Definition</u>
a\$	String	Variable type containing up to 32767 characters.
a	Real	A six byte floating point variable with an accuracy to 11 digits. (In scientific notation, exponents of up to 154 are permitted.)
a%	Integer	An integer variable containing a whole number in the range of -2147483648 to +2147483647. Integer variables require 4 bytes of storage.
a!	Boolean	Boolean variables can only contain the values of 0 (FALSE) or -1 (TRUE). They require 2 bytes for storage.

Variables may be stored in binary (base 2), octal (base 8), decimal (base 10), or hexadecimal (base 16) formats. Variables written without a prefix, as shown below, are considered to be decimal values.

x = 10

Binary values are stored by assigning the value as:

x = &X1110

Octal values are preceded by &O.

x = &O8

Hexidecimal values are preceded by &H.

x = &HFF

Mathematical Functions

The mathematical operations of GFA BASIC are expressed by the following symbols:

<u>Symbol</u>	<u>Meaning</u>
+	Addition (sum of two numbers)
-	Subtraction (difference between two numbers)
*	Multiplication
/	Division
DIV	Integer division of numbers. (Converts to a whole number.)
MOD	Gives integer remainder of integer division.
^	Exponential (Raising a number to a power)

When a mathematical operation is performed, or an equation is evaluated, the functions are dealt with in the following order:

<u>Function</u>	<u>Description</u>
(Parenthesis have the highest order of priority and are evaluated first. They may be used to change an undesirable order of precedence.
^	Exponential.
+ or -	Plus or minus signs when used as a prefix for a numerical value.
* or /	Multiplication or division.
DIV MOD	Integer division and modulation
+ or -	Addition and subtraction
=	Comparisons
<> < > <= >=	
NOT AND OR XOR IMP EQV	Logical functions

Hierarchical File Structures

The Atari ST uses tree structured directories. This means that its directories branch out from each other. Each directory can contain files and subdirectories that can, in turn, contain files and more subdirectories. This system is usually referred to as the hierarchical file system.

The first level of this system is the folder which may be encountered on the GEM desktop. This is the *Root* directory.

For example, the root directory for disk drive A is called "A:\\". Directories contained within the root directory are referred to as *child* directories. Child directories within the same root directory are said to stem from the same *parent* directory.

While a directory may have many child directories, it may only have one parent directory.

When operating within the hierarchical file structure, each parent and child is separated by the backslash (\) symbol. For example:

A:\PARENT1\CHILD1\CHILD2\TEST.DOC

provides a path for access to a file named TEST.DOC, stored in a folder called CHILD2. The CHILD2 folder is stored within a folder named CHILD1, which in turn is stored within a folder called PARENT1. PARENT1 is on the root directory.

Although at first glance this system may seem rather confusing, it will become clear and quite logical with use.

Chapter 1

The GFA Editor/ Interpreter



Chapter 1: The GFA Editor/Interpreter

To write a GFA BASIC program, you must first place program statements in memory. These program statements are executed by the GFA BASIC interpreter.

Program statements are entered by using the GFA Editor/Interpreter, which is an extremely powerful programming tool.

Although not a GEM editor, the GFA Editor/Interpreter provides a potent arsenal of weapons for your quest to push the limits of your Atari ST. The Editor/Interpreter supports block operations, search and replace functions, and also checks for typing errors in commands as you enter program statements.

Let's begin examining the Editor/Interpreter at the very beginning. Even if you have some experience with GFA BASIC, read through this section. You might discover something new. Besides, before we can begin to study the GFA BASIC language, you must understand how to function within the GFA programming environment.

Place your GFA BASIC system disk in drive A, and use the mouse to select GFABASIC.PRG. Your screen should look like Figure 1-1.

Figure 1-1. The GFA BASIC Editor/Interpreter

The screenshot shows a window titled "GFA BASIC Editor/Interpreter". The menu bar includes "File", "Edit", "Format", "Tools", "Help", and "Run". The main window displays the following BASIC code:

```
| Save | Save, R | Quit | New | 181k Sta|Replace| Pg up |Text 16|Direct | Run |
| Load | Merge | List | Block | 181k End| Find | Pg down|Insert | Flip | Test |
DIM s%(32255/4)
rate:=2
DEFFILL 2
GRAPHMODE 1
SETCOLOR 0,3,4,5
SETCOLOR 15,7,7,7
COLOR 8
PCIRCLE 100,100,20
GET 75,75,125,125,a$
CLS
DEFFILL 3
PCIRCLE 100,100,20
GET 75,75,125,125,b$
CLS
DEPMOUSE 5
a%:=XBIOS(3)
b%:=VARPTR(s%(0))+255 AND &HFFFF00
SGET h$
x%:=100
y%:=100
dx%:=0
dy%:=0
REPEAT
```

One of the unique features of the GFA BASIC Editor/Interpreter is that even if you are in low resolution (color) mode, you still will be able to edit an 80 column screen. The mouse will only be active on the left half of the screen, and the file selector box will appear on the left side rather than in the middle, where it normally appears.

Before we examine each of the twenty Editor/Interpreter functions on the command line, let's take a look at the structure of a GFA program line, and how it's entered into memory.

Program lines in GFA BASIC may contain only one command per line, but each line may contain up to 255 characters. Since the screen can only display 80 columns, any characters entered in excess of 80 will cause the line to scroll to the left. The left most characters may scroll off the screen, but the line will not be altered in any way. When you finish editing the line, for instance by pressing <Return>, only the first 79 characters in the line will be displayed.

Due to the storage characteristics of GFA BASIC, it is possible for a line with less than 255 characters to use 255 bytes, the maximum number of bytes permitted for each line. If this occurs, the message "LINE TOO LONG" will be displayed. It will then be necessary to alter the line to make it acceptable to the Editor/Interpreter. Program lines are entered from the keyboard, and special characters not assigned a key may be entered by using the <Alternate> or <Control> keys. A program line may be terminated by pressing <Return> or by using the cursor keys to move off the line.

When a line is exited, the syntax of that line is checked. If any error was made, the message "Syntax Error" will be displayed on the command line, and you will not be permitted to leave the line. Correct any errors and try again.

After a line has been successfully entered, the editor takes control for any special formatting needed. GFA BASIC programs use indentation to make the program structure more distinct. This indentation is handled automatically by the editor.

Also, if a command was abbreviated, the editor substitutes the full command for the abbreviation. (See Chapter 2, The GFA BASIC Language, for all command abbreviations.)

The default display of text in the GFA BASIC Editor/Interpreter is with every word shown with initial caps. A more readable listing may be obtained by pressing the <Esc> key to switch to the display screen. Then type DEFLIST 0 and press <Return>.

Now type ED and press <Return>.

When you return to the Editor/Interpreter screen, all BASIC key words will be all capitol letters, and all variables will be in lower case letters. You may find this listing display easier to read. This is the listing format used for the sample programs in this book.

If you prefer initial capitol letters on keywords and variable names, as the GFA BASIC Editor/Interpreter appears when first installed, enter DEFLIST 1 to restore the default setting.

The Command Line

Notice the two command lines at the top of the screen, as shown in Figure 1-1. Each of the twenty editor functions (ten on each line), may be accessed by using the mouse pointer, by pressing the function keys, or a <Shift> function key combination. (If you are in low resolution mode, it will be necessary to use the function keys to access all commands on the right half of the screen.)

Save

The first function on the upper command line is the *Save* function. This function may be selected by pressing the <Shift> <F1> keys or by moving the mouse selector to highlight *Save* and clicking the left mouse button. The GEM file selector box will be displayed, and a file name to use for storing the file on your disk may be used. The default file name offered by the file selector is the same as the last name used to save the file. The previous version will be saved as *filename.BAK* and replaced by the newer version.

As with any, GEM file selector, the hierarchical file system may be used for directing the file to be saved in a folder, or on another disk.

The file will be saved with a .BAS extension, unless another extension is specified. Files are saved in a tokenized form. That is, in a semicomplied form. These files are shorter than a pure ASCII file, and enable the interpreter to execute the code more efficiently. Files saved with the *Save* function may only be loaded into the GFA BASIC Editor/Interpreter, or run with GFABASICRO.PRG, the run only module supplied with GFA BASIC.

Load

The command located under *Save* is the *Load* command, on the lower line of the command line. This command may be accessed either by pointing to it with the mouse pointer and clicking the left mouse button, or by pressing the <F1> key.

Load displays the standard GEM file selector window, and permits a file, previously stored with the *Save* command to be loaded into memory. The search path is preset to “*.BAS”, so only files with the .BAS extension are displayed.

Only GFA BASIC programs, (usually with the .BAS extension on the filename) may be placed in memory using this command.

Save,A

The *Save,A* command permits the programmer to save a file as a text file, in ASCII form, and may be selected either by pointing with the mouse pointer and clicking the left mouse button, or by pressing <Shift><F2>.

The file is saved preserving the DEFLIST mode selected by the programmer.

Although this command may not appear useful at first, it is in reality quite handy. Files saved as ASCII files may be merged with Word Processor documents, appended to GFA BASIC files within the Editor/Interpreter, or even uploaded as Text files to telecommunication information services.

This feature is actually very important. It permits you to construct libraries of useful routines, then merge the needed routines into a program. This will eliminate the need to rewrite commonly used routines every time you need them.

Also, if your disk should be corrupted, as sometimes happens, you may be able to load, using the *Merge* command, and correct an ASCII file, even when your .BAS file is damaged beyond the ability of the Editor/Interpreter to install it.

When the *Save,A* command is selected, the .LST suffix is automatically appended to the filename, unless another extender is specified.

Merge

The *Merge* command may be selected by pointing with the mouse pointer and clicking the left mouse button, or by pressing the <F2> key.

Merge permits ASCII, or files previously saved using the *Save,A* command to be merged with any program in memory. The default search path is “*.LST”.

The file selected will be added to the program segment in memory beginning with the line immediately above the cursor location.

Should a syntax error exist in the merged program, the symbol ‘==>’ appears in front of the line containing the error. This is most likely to happen when attempting to convert a program written in another form of BASIC, such as an ST BASIC program, into a GFA BASIC program.

Quit

The *Quit* command, selected by pointing to the command with the mouse pointer and clicking the left mouse button, or by pressing the <Shift><F3> key, permits the programmer to exit the GFA BASIC Editor/Interpreter and return to the GEM Desktop. An Alert box will be displayed, requesting confirmation of your intentions.

Any program information not previously saved will be lost.

Llist

The *Llist* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F3> key, will send a listing of the program in memory to an attached and ready printer.

No check is made to confirm that a printer is attached and ready to receive data from the computer, and the print out may only be interrupted by turning off the printer.

If the printer is turned off, or no printer is available, control will return to the Editor/Interpreter after about 30 seconds.

New

The *New* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <Shift><F4> key, clears all program information from memory, effectively leaving a blank page in the editor. Any information not previously saved will be lost.

An Alert box will ask for confirmation before completing this command, because of the finality of the operation.

Block

The *Block* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F4> key, works in conjunction with the *Blk Sta* and *Blk End* command, which are discussed next.

Block allows the selection of various functions which will be performed on a previously defined block within the program listing. If one or both marks necessary for defining a block are missing when this command is selected, the error message "BLOCK ????" will appear. This message may be removed by pressing the up or down arrow keys, or by clicking the mouse button.

If a block has been properly defined, a selection menu will appear instead of the normal command line, and a choice may be made by pressing the appropriate key, or by pointing and clicking with the mouse.

The *Copy* command, which may be selected by pressing the <C> key, copies the defined block to the area immediately above the cursor location. The cursor must be located outside the defined block.

The *Move* command, or the <M> key, moves the defined block to the area beginning one line above the cursor. This again is only possible when the cursor is located outside of the defined block. The block is deleted from its previous position.

The *Write* command, or the <W> key, writes the block to a disk file as an ASCII file, with the .LST suffix on the filename. The standard GEM file selector box is used. This operation is handy for creating a library of useful routines to be merged with other programs.

The *Llist* command, or the <L> key, sends the defined block to an attached printer. As with *Llist* from the main command line, no check is made to see if a printer is attached. If this command is selected without having the printer attached or ready to receive data, a time out will occur after about 30 seconds, and then control will be returned to the Editor/Interpreter.

The *Start*, <S>, command moves the cursor to the beginning of a block. This is handy when testing a long program because you can return to an area where you are interested in making changes. Define the block, run the program, break out, or allow the program to end, then select *Block* and *Start* to return to that area.

The *End*, <E>, command is similar to the *Start* command, except *End* places the cursor at the end of a defined block.

The *Delete* command, which may also be selected by pressing the <Control> and <D> keys at the same time, removes a defined block of text. This operation cannot be undone by pressing the <Undo> key, so be sure you mean it before selecting this operation.

The *Hide*, <H>, command removes the block markers, and returns the displayed listing to normal. (As a block of text is defined, the background is shaded to visually mark the block for the convenience of the programmer.)

Blk Sta

The *Blk Sta* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <Shift><F5> key, marks the beginning of a block of text. This command works in conjunction with *Blk End* and the *Block* commands.

As the cursor is moved with the cursor keys, the background of the defined block is shaded for the convenience of the programmer.

Blk End

The *Blk End* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F5> key, marks the end of a defined block of text. This command works in conjunction with the *Blk Sta* and *Block* command, as previously noted.

The area between the mark placed by *Blk Sta* and the mark placed by *Blk End* is shaded for the convenience of the programmer.

Replace

The *Replace* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <Shift><F6> key, is an extremely powerful command, and permits a specified word or group of words to be searched for and selectively replaced by another word or group of words. A string of up to 60 characters may be specified as the search string or replacement string.

Replace is case sensitive, that is, the word or group of words being searched for must exactly match the specified string in order to be found. When the word, or group of words is found, the cursor will be positioned at the start of the group, and it may be replaced by pressing <Control><R>. If the programmer chooses not to replace the item, <Control><F> may be pressed, and the next match will be searched for.

If <Control><R> is pressed, the word or group of words is deleted, and the replacement string is inserted. Pressing <Control><F> causes the Editor to search for the next occurrence of the search string.

It's a good idea to press <Control><Home> to move the cursor to the beginning of the program before conducting a search and/or replace operation to insure that all occurrences of the search string are found.

If the replacement option, <Control><R>, is selected in error, <Undo> will restore the item as it was in the original line.

When the search fails, the cursor will be positioned at the end of the program listing, and a message indicating the search string was not found will be displayed on the command line.

Find

The *Find* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F6> key, is another powerful command, and permits a specified word or group of words to be searched for within a program listing. A string of up to 60 characters may be specified as the search string.

Find is also case sensitive, that is, the word or group of words being searched for must exactly match the specified string, including capitol letters, in order to be found.

When the word, or group of words is found, the cursor will be positioned at the start of the group. To search for the next occurrence of the search string, press <Control><F>.

It's a good idea to press <Control><Home> to move the cursor to the beginning of the program before conducting a search operation to insure that all occurrences of the search string are found.

Pg up

The *Pg up* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <Shift><F7> key, moves the cursor to the Home position of the page which precedes the page currently displayed. This command may also be called by pressing <Control> and the up arrow key.

Pg down

The *Pg down* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F7> key, moves the cursor to the Home position of the page which follows the page currently displayed. This command may also be called by pressing <Control> and the down arrow key.

Text 16 Text 8

Monochrome (high resolution) monitor programmers have a choice of two text sizes by selecting *TEXT 8* or *TEXT 16* from the command line, either by pointing and clicking with the mouse, or by using the function keys.

Color monitor programmers will not see this function on the command line. Any empty block will occupy the *TEXT* function position, as only *TEXT 16* is available on Color monitors.

By selecting *TEXT 16*, 23 program lines may be displayed. *TEXT 8* permits 48 lines of text, displayed in a reduced character size, allowing somewhat easier editing of program files.

This command is a *toggle* function, in other words, the two display modes may be chosen by selecting this option and switching between the them.

Insert Overwrt

The *Insert* and *Overwrt* commands, may be selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F8> key.

Insert causes any keystrokes to be inserted into the text, pushing any existing text ahead of the cursor.

Ovrwrt, (Overwrite), causes any key strokes to overwrite, or replace existing text under the cursor position.

Direct

The *Direct* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <Shift><F9> key, switches to the output screen, and permits commands to be entered in *Direct mode*, where the commands are executed immediately.

This is handy for checking the effect of a command, or for calling a subroutine for testing purposes.

Flip

The *Flip* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F9> key, is another method of switching to the output screen.

Run

The *Run* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <Shift><F10> key, causes the program currently in memory to be executed.

Test

The *Test* command, selected by pointing to the command with the mouse pointer and clicking with the left mouse button, or by pressing the <F10> key, checks all loops in the program, and notifies the programmer of any errors.

Keyboard Commands

The power of the GFA Editor/Interpreter extends to the use of keyboard commands, many of which effectively replace command line functions, permitting ease of movement through the program listing.

<u>Key Strokes</u>	<u>Function</u>
<Control><Left Arrow>	Moves cursor to the beginning of the line.
<Left Arrow>	Moves cursor left one character.
<Control><Right Arrow>	Moves cursor to the end of the line.
<Right Arrow>	Moves cursor right one character.
<Control><Up Arrow>	Moves cursor one page back.
<Up Arrow>	Moves cursor up one line.
<Control><Down Arrow>	Moves cursor ahead one page.
<Down Arrow>	Moves cursor down one line.
<Control><Tab>	Moves cursor left one tab position.
<Tab>	Moves cursor right one tab position.
<Return>	Moves cursor down one line.
<Control><Home>	Moves cursor to beginning of program.
<Home>	Moves cursor to beginning of the currently displayed page.
<Control><Z>	Moves the cursor to the end of the program.
<Backspace>	Moves the cursor left one character and deletes that character.
<Delete>	The character under the cursor is deleted.
<Control><Delete>	The entire line the cursor is located on is deleted.
<Insert>	Moves all lines beginning with the line on which the cursor is located down one line, and allows text to be inserted into the listing.

- | | |
|---------------------------------|--|
| <Undo> | Reverses last changes made to a program line, restoring the line to its original condition, provided that the cursor has not been moved off of the line. |
| <Control><F> | Finds next occurrence of specified search string. |
| <Control><R> | Replaces search string with specified replacement string. |

Chapter 2

The GFA BASIC Language



Chapter 2: The GFA BASIC Language

Before any language can be used, it must be understood. Some languages, such as Assembly language, contain only a limited number of words which must be configured to create routines which in turn accomplish the desired functions within a program. Fortunately, GFA BASIC has many built-in functions, making the life of a programmer much easier.

This chapter contains every word in GFA BASIC, Version 2.0, listed in alphabetical order. A few terms must be understood so these words can be discussed.

A word in any version of BASIC is generally accepted to be any word found in a BASIC program, or used to control a BASIC program. Words can be broken down into five sub-categories:

Commands are BASIC words which tell the computer to execute some action, or do something with a program. For instance RUN, LIST, SAVE, LOAD, and so forth are all BASIC commands.

Statements are words which are used inside programs.

Functions are words which perform relatively complicated actions, such as finding trigonometric values and square roots.

Operators are words and characters which perform specific logical operations. (AND, NOT, NOR, +, -, and so forth).

System variable, although not generally thought to be BASIC words, are included in this listing, as they are very much a part of the language. These are words which return a value defined by the operating system or by GFA BASIC.

This chapter should be used as a general reference to the GFA BASIC language. Occasionally, words may be listed as Functions in the lexicon listing rather than as commands shown in the GFA BASIC manual. This minor deviation was done to allow a broader explanation of the language.

The words included in GFA BASIC may be broken down into groups as shown in the following table.

<u>Constants</u>	<u>Mathematical Functions</u>	<u>Program Control Statements</u>
ADDRIN	ABS	DEFFN
ADDROUT	ADD	DO...LOOP
BASEPAGE	ATN	EXIT IF
CONTRL	COS	FN
FALSE	DEC	FOR...NEXT
GB	DIV	GOSUB
GCTRL	EVEN	GOTO
GINTIN	EXP	IF..[THEN]...[ELSE]...ENDIF
GINTOUT	FIX	LOCAL
HIMEM	FRAC	PROCEDURE
INTIN	INC	REPEAT...UNTIL
INTOUT	INT	RETURN
PI	LOG	WHILE...WEND
PTSin	LOG10	
PTSout	MUL	
TRUE	ODD	
VDIBASE	RANDOM	
WINDTAB	RND	
	SGN	
	SIN	
	SQR	
	SUB	
	TRUNC	
	VOID	
<u>Operators</u>		<u>String Handling Commands</u>
()		&H
*		&O
+		&X
+		ARRAYFILL
+ (String)		ARRPTR
-		ASC
-		BIN\$
/		CHR\$
<		CVD
<= <=		CVF
>		CVI
=		CVL
==		CVS
>		DIM
>= >=		FRE(0)
^		HEX\$
AND		INSTR
DIV		LEFT\$
EQV		LEN
IMP		LSET
MOD		MAX
NOT		MID\$
OR		MIN
XOR		MKD\$
	REM or ' or !	MKF\$
	RUN	MKI\$
	STOP	MKL\$
	SYSTEM	MKS\$
	TROFF	
	TRON	
	<u>System Commands</u>	
	CLEAR	
	CLR	
	CONT	
	DEFLIST 0	
	DEFLIST 1	
	EDIT	
	END	
	ERASE	
	LIST	
	LIST "file.lst"	
	LLIST	
	NEW	
	QUIT	
	REM or ' or !	
	RUN	
	STOP	
	SYSTEM	
	TROFF	
	TRON	

OCT\$
 OPTION BASE 0
 OPTION BASE 1
 RIGHT\$
 RSET
 SPACE\$
 STR\$
 STRING\$
 SWAP
 TYPE
 UPPER\$
 VAL
 VARPTR

DATA Handling

DATA
 LET
 READ
 RESTORE

Input and Output

CRSCOL
 CRSLIN
 DEFNUM
 FORM
 HARDCOPY
 INKEY\$
 INP
 INP?
 INPUT
 INPUT
 INPUT
 INPUT\$
 LINE
 LPOS
 LPRINT
 OUT
 OUT?
 POS
 PRINT
 PRINT AT
 PRINT SPC
 PRINT TAB
 PRINT USING
 PRINT#
 WRITE
 WRITE#

Sequential and Random File Handling

CLOSE
 EOF
 FIELD
 FORM
 GET
 INP
 INPUT
 INPUT
 INPUT#
 INPUT\$
 LEN

LINE
 LOC
 LOF
 OPEN
 OUT
 PRINT#
 PUT
 RELSEEK
 SEEK
 WRITE#

Disk Commands

BGET
 BLOAD
 BPUT
 BSAVE
 CHAIN
 CHDIR
 CHDRIVE
 DFREE
 DIR
 DIR\$
 EXEC
 EXIST
 FILES
 FILESELECT
 KILL
 LIST
 LOAD
 MKDIR
 NAME
 PSAVE

RMDIR
 SAVE

Graphics Commands

BITBLT
 BMOVE
 BOX
 CIRCLE
 CLS
 COLOR
 DEFFILL
 DEFLINE
 DEFMARK
 DEFTEXT
 DRAW
 ELLIPSE
 FILL
 GET
 GRAPHMODE
 LINE
 PBOX
 PCIRCLE
 PELLIPSE
 PLOT
 POINT
 POLYFILL
 POLYLINE
 POLYMARK
 PRBOX
 PUT
 RBOX
 SETCOLOR
 SGET
 SPRITE
 SPUT
 TEXT
 VSYNC

Mouse Commands

DEFMOUSE
 HIDEM
 MOUSE
 MOUSEK
 MOUSEX
 MOUSEY
 SHOWM

Sound

SOUND
WAVE

LPOKE

MONITOR

PEEK

POKE

RESERVE

SDPOKE

SLPOKE

SPOKE

VDISYS

XBIOS

Timer Functions

DATE\$
PAUSE
SETTIME
TIMER
TIME\$

Break and Error

ERR
ERROR
FATAL
ON BREAK
ON BREAK CONT
ON BREAK GOSUB
ON ERROR
ON ERROR GOSUB
RESUME
RESUME NEXT

Windows and Menus

ALERT
CLEARW
CLOSEW
CLOSEW 0
FULLW
INFOW
MENU
MENU KILL
MENU OFF
MENU x,y
MENU()
ON MENU
ON MENU BUTTON
ON MENU GOSUB
ON MENU IBOX
ON MENU KEY
GOSUB
ON MENU MESSAGE
GOSUB
ON MENU OBOX
OPENW
OPENW 0
TITLEW

System Level Commands

BIOS
CALL
DPEEK
DPOKE
GEMDOS
GEMSYS
LPEEK

The following section presents each word in GFA BASIC in alphabetical order. The GFA BASIC word to be explained appears as a heading for each section, along with a descriptor for the part of the language, for easy indexing.

Next, a sample of the format for using the word is shown, with a description of the word and a list of all required and optional parameters associated with that word.

In most cases, a sample program segment is shown, demonstrating the use of that particular word. The sample code is usually not the most efficient way to achieve a desired result, but represents the simplest, most easily understood explanation. In cases where a simple example will not suffice, it is hoped that enough explanation is included so that you can understand the concept of the command.

* Operator

Syntax: *Variable A * Variable B*

This symbol is used to specify the multiplication operator. It provides the product of the multiplication of two integer or real variables or numbers.

Example:

PRINT 3*2

Related Words:

MUL

+
Operator**Syntax:** *Variable A + Variable B*

This is the addition operator. It provides the sum of two integer or real variables or numbers.

Example:

PRINT 2+3

Related Words:ADD

-
Operator**Syntax:** *Variable A — Variable B*

This symbol is the subtraction operator and provides the difference between two integer or real variables.

Example:

PRINT A-B

Related Words:

SUB

/ Operator

Syntax: *Variable A / Variable B*

This is the division symbol and provides the result of the division of two integer or real variables.

Example:

```
PRINT A/B
```

Related Words:

DIV, MOD

< Operator

Syntax: *Variable A < Variable B*

This is the 'less than' relational operator. It is used to compare the numeric values of two integer or real variables. It is generally used in conjunction with IF...THEN, WHILE...WEND, or REPEAT...UNTIL statements.

Example:

```
a=0  
WHILE a<=10  
    INC a  
WEND  
PRINT a
```

<=
Operator

Syntax: *Variable A <= Variable B*

This symbol means 'less than or equal to'. This relational operator compares the numeric value of two integer or real variables. Usually used in conjunction with IF...THEN, WHILE...WEND, or REPEAT...UNTIL statements.

Example:

```
a=0  
WHILE a<=10  
    INC a  
WEND  
PRINT a
```

<>
Operator

Syntax: *Variable A <> Variable B*

This is the symbol for 'not equal to', or inequality. This relational operator compares the numeric value of two integer or real variables. Usually used in conjunction with IF...THEN, WHILE...WEND, or REPEAT...UNTIL statements.

Example:

```
a=0  
WHILE a<>10  
    INC a  
WEND  
PRINT a
```

=**Operator****Syntax:** *Variable A = Variable B*

This is the symbol for the equal sign. This relational operator compares the numeric value of two integer or real variables. It may also be used to assign a value to a variable.

Example:

```
A=1  
B=2  
C=3  
D=A+B+C
```

>**Operator****Syntax:** *Variable A > Variable B*

This is the symbol for 'greater than'. This relational operator compares the numeric value of two integer or real variables. Usually used in conjunction with IF...THEN, WHILE...WEND, or REPEAT...UNTIL statements.

Example:

```
a=20  
WHILE a>10  
  DEC a  
  PRINT a  
WEND
```

>=
Operator**Syntax:** *Variable A >= Variable B*

This is the symbol for greater than or equal to. This relational operator compares the numeric value of two integer or real variables. Usually used in conjunction with IF...THEN, WHILE...WEND, or REPEAT...UNTIL statements.

Example:

```
a=20
WHILE a>=10
  DEC a
  PRINT a
WEND
```

^
Operator**Syntax:** *Variable A ^ power*

The ^ (caret) is used as a symbol for arithmetically computing the value of a base number raised to a specified power.

Example:

```
PRINT 3^2
```

ABS

Function

Syntax:**PRINT ABS(x)**

ABS returns the value of an integer or real number, numerical expression, or variable without a positive (+) or negative (-) sign. The absolute value of an expression is always greater than or equal to zero.

Example:

```
a=-1234  
b=1234  
PRINT ABS(a)  
PRINT ABS(b)  
PRINT ABS(&Haef)  
PRINT ABS(3*(-7))
```

ADD

Function

Syntax:**ADD *variable a,variable b***

The ADD function performs the same operation as the + operator. The advantage of using ADD is that the speed of execution is increased significantly. This example demonstrates the advantage of using ADD.

Example:

```
t=TIMER  
FOR ctr%=1 TO 1000  
    ADD a%,5  
NEXT ctr%  
PRINT (TIMER-t)/200  
a%=0  
t=TIMER  
FOR ctr%=1 TO 1000  
    a%=a%+5  
NEXT ctr%  
PRINT (TIMER-t)/200
```

Related Words:

+

ADDRIN
System Variable

Contains the address of the AES (Application Environment Services) address input block. (Refer to GEMSYS.)

ADDROUT
System Variable

Contains the address of the AES (Application Environment Services) address output block. (Refer to GEMSYS.)

ALERT
Function

Syntax: **ALERT icon, text string, button, button text, variable a**

Abbreviation: **A**

An Alert box is a part of GEM, and may be used to warn a program user of a potential problem, request input from the user, or give hints.

Parameters:

Icon This parameter is a number (0-3), which causes an icon to be displayed within the Alert box.

- 0 No icon
- 1 !(exclamation point)
- 2 ?(question mark)
- 3 Stop Sign

Text_String The next parameter is a character string expression, which contains the main text to be displayed within the Alert box. The text may be broken up into a maximum of four lines by separating the text with the ‘!’ character.

Button Next is the parameter which specifies which of three possible buttons is to be highlighted as the default button. This button is defined as the choice selected if the user simply presses <Return>. Only one button may be highlighted.

- 0 No button highlighted.
- 1 First button highlighted.
- 2 Second button highlighted.
- 3 Third button highlighted.

Button_text This parameter is a character string expression which contains the text to be displayed within the exit buttons. The text of each button is limited to no more than 8 characters.

Variable_a This parameter returns the number of the exit button selected by the user.

Example:

```
text_string$="This is an Alert Box."  
ALERT 1,text_string$,1,"OK",variable_a
```

AND Operator

Syntax: *condition AND condition*

This is a logical operator. AND does not evaluate conditions, but logically compares the results of conditions. Usually used as a logical operator in conjunction with IF...THEN, WHILE...WEND, and REPEAT...UNTIL statements. Both conditions must be evaluated as true for the appropriate action to occur.

Condition AND Condition = Result

True (-1)	AND	True (-1)	=	True (-1)
True (-1)	AND	False (0)	=	False (0)
False (0)	AND	True (-1)	=	False (0)
False (0)	AND	False (0)	=	False (0)

AND may also be used as a bit-wise comparator, as in Boolean Algebra, in evaluating the bit settings of a binary number.

	<u>Decimal Value</u>	<u>Binary Value</u>
	3	0011
AND		AND
	5	0101
Result	1	0001

Example:

```
a%=0  
b%=0  
REPEAT  
    INC a%  
    INC b%  
    PRINT a%,b%  
UNTIL a%=10 AND b%=10
```

or;

```
a%=3  
POKE VARPTR(a%),a% AND 5
```

Related Words
OR, XOR, NOT, IMP, EQV, ^

ARRAYFILL
Function

Syntax: **ARRAYFILL** *field()*,*n*

Abbreviation: AR

This statement assigns a value to all elements within an array. All elements within the array field are filled with the numeric value, n.

Parameters:

field Any numeric or boolean array.

n Can be a number or numeric variable.

Example:

```
DIM a%(10)
FOR ctr%=0 to 10
    PRINT a%(ctr%)
NEXT ctr%
ARRAYFILL a%(),1
FOR ctr%=0 TO 10
    PRINT a%(ctr%)
NEXT ctr%
```

ARRPTR

System variable

Syntax: **ARRPTR(variable)**

ARRPTR determines the address of the descriptor of a string or array variable. The descriptor is 6 bytes long. The first fours bytes contain the address of the array in memory. The last two bytes contain the number of dimensions for the array.

Parameters:

variable Any string or numeric variable.

Example:

```
a$="String variable"
PRINT ARR PTR(a$)
PRINT LPEEK(ARR PTR(a$)) ! Start Address
PRINT DPEEK(ARR PTR(a$)+4) ! Length of array
```

Related Words

VARPTR, LEN

ASC**Function****Syntax:** **ASC (string_variable)**

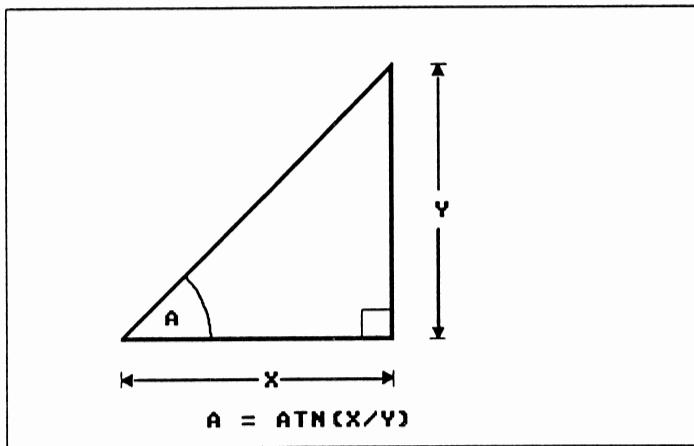
This function determines the ASCII code of the first character in a string variable.

Parameters:**string_variable** Any string variable**Example:**

```
a$="Another string variable"  
PRINT ASC("A")  
PRINT ASC("A string variable")  
PRINT ASC(a$)
```

ATN**Function****Syntax:** **ATN(r)**

The ATN function returns the trigonometric arc tangent of an angle in radians. One radian equals about 57 degrees. Arctangent is defined as the angle which corresponds to a specific ratio created by dividing the length of the side opposite the angle by the length of the side adjacent to the angle. Literally, ATN is the Arc (angle) of the TaNgent (ratio).

Figure 2-1 Arc Tangent

To convert this value from radians to degrees, multiply the value returned by ATN times 180/PI.

Parameters:

r Any numeric or arithmetic expression.

Example:

```
INPUT "Angle to evaluate:";angle%
r%=ATN(angle%)
PRINT "The Arc tangent of ";angle%;" is ";r%;" radians"
d%=r%*(180/PI)
PRINT "or ";d%;" degrees."
```

Related Words:

COS, SIN, TAN

BASEPAGE**System Variable****Syntax:****BASEPAGE**

Returns the address of the 256 byte basepage assigned to GFA BASIC by GEMDOS. The first 128 bytes contain essential information needed by the operating system. A careless POKE here can cause a disaster. The second 128 byte segment contains the command line which can be used by the EXEC command from GFA BASIC. DIR and DIR\$ also use a portion (bytes 128-172) of this second 128 byte segment. Small machine language routines can also be safely placed here. Refer to Table 2-2 on the next page for more information.

Table 2-2 Base page format block.

<u>Offset</u>		<u>Description</u>
0	\$00	Base address of Transient Program area.
4	\$04	End of Transient Program area + 1.
8	\$08	Base address of text.
12	\$0C	Length of text.
16	\$10	Base Address of Init data.
20	\$14	Length of Data
24	\$18	Base address of Block Storage Segment uninit data.
28	\$1C	Length of Block Storage Segment unit data.
32	\$20	Length of free memory after Block Storage Segment.
36	\$24	Drive from which program loaded.
37	\$25	Reserved by XBIOS.
56	\$38	2nd parsed File Control Block.
92	\$5C	1st parsed File Control Block. 1st and 2nd File Control Blocks make up the Command line as set by the Console Command Processor.
128	\$80	Command tail and default Memory Access buffer.
.	.	
.	.	
.	.	This area may be used by programs.
.	.	
.	.	
256	\$FF	End of Base page.

BGET
Statement

Syntax: **BGET [#] channel number, addr, bytes**

Reads from a specified data channel and stores the information in memory. Unlike BLOAD, data may be read into several different areas of memory from a file.

Parameters:

[#] channel number The channel number set by OPEN.

addr Location in memory where data will be stored.

bytes Number of bytes of data to read and store.

Example:

```
OPEN "O",#1,"MYDATA.TXT"
BGET #1,&H1234,126
CLOSE #1
```

Note! This is not a functioning program, but demonstrates the sequence of commands to BGET 126 bytes of data and store it in memory location &H1234.

Related Words:

BPUT, BSAVE, BLOAD

BIN\$
Function**Syntax:** **BIN\$(*x*)**

This function converts the value of *x* to a character string containing the binary value of *x*.

Parameters:

x Any integer between -214783648 and +214783647 in normal decimal, hexidecimal (&H), octal (&O), or binary (&X) form.

Example:

```
a=12  
b=&H45  
PRINT BIN$(a)  
PRINT BIN$(b)  
PRINT BIN$(435)
```

Related Words:
HEX\$, OCT\$, STR\$

BIOS
Function**Syntax:****BIOS(*n*, *parameter_list*)**

Calls the BIOS (Basic Input/Output System) functions, similar to using TRAP #13 from Assembly Language. These functions return a 32-bit number. Some commands expect 16-bit words passed as parameters, others 32-bit long words. 32-bit words passed to a BIOS routine must be prefaced with 'L'. 16-bit words may use the optional 'W' preface. Later chapters have several examples of using BIOS.

Parameters:

n ID number of BIOS routine to be called.

parameter_list List of parameters required by the BIOS function being called. See Appendix B for a list of BIOS functions.

Example:

```
drive_map% = BIOS(&HOA) !Bitmap of installed disk drives
```

Related Words:
GEMDOS, XBIOS

BITBLT
Function**Format:****BITBLT *smfdb%()*,*dmfdb%()*,*p%()***

BITBLT is a raster copying function, similar to the GET/PUT combinations. BITBLT is more flexible, but consequently more difficult to use properly. Although generally associated with transferring Graphics information on the screen, it may be used to move any area of memory.

Parameters:**Source Memory Form Definition Block**

smfdb%(0) Even address of area which contains the data to be moved.

smfdb%(1) Raster width (in pixels)
640 High Resolution (Monochrome)
320 Medium and Low resolution (Color)

smfdb%(2) Raster height (in pixels)
400 for High resolution (Monochrome)
200 for Medium and Low resolution (Color)

smfdb%(3) Raster width is 16-bit words. This may be calculated by using the expression:

$$\text{INT}((\text{smfdb}%(1)+15)/16)$$

Adding 15 to the value in smfdb%(1) ensures an even boundary.

smfdb%(4) Always zero.

smfdb%(5) Number of bit planes.
1 = Monochrome monitor
2 or 4 = Color monitor

smfdb%(6—8) Reserved for future use.

Destination Memory Form Definition Block

dmfdb%(0) Even address of area to which the data is to be moved.

dmfdb%(1) Raster width (in pixels)
640 High Resolution (Monochrome)
320 Medium and Low resolution (Color)

dmfdb%(2) Raster height (in pixels)
400 for High resolution (Monochrome)
200 for Medium and Low resolution (Color)

dmfdb%(3) Raster width is 16-bit words. This may be calculated by using the expression:

$$\text{INT}((\text{dmfdb}%(1)+15)/16)$$

dmfdb%(4) Always zero.

dmfdb%(5) Number of bit planes.

1 = Monochrome monitor
2 or 4 = Color monitor

dmfdb%(6—8) Reserved for future use.

Memory Form Definition Block

<u>Offset</u>	<u>Contents</u>
0 \$00	32-bit address
4 \$04	Width of Raster
8 \$08	Height of Raster
12 \$0C	Word Width (Pixels/16)
16 \$10	Format flag 1= standard 0= device specific (Always zero for Atari ST)
20 \$14	Number of planes in raster area
24 \$18	...
28 \$1C	Three reserved words
32 \$20	...

- p%(0)* Source rectangle point x0
- p%(1)* Source rectangle point y0
- p%(2)* Source rectangle point x1
- p%(3)* Source rectangle point y1
- p%(4)* Destination rectangle point x0
- p%(5)* Destination rectangle point y0
- p%(6)* Destination rectangle point x1
- p%(7)* Destination rectangle point y1

p%(8) Mode:

- | | |
|----|--------------------------------|
| 0 | delete |
| 1 | source AND destination |
| 2 | source AND (NOT destination) |
| 3 | source (overwrite) |
| 4 | (NOT source) AND destination |
| 5 | destination (do nothing) |
| 6 | source XOR destination |
| 7 | source OR destination |
| 8 | NOT (source OR destination) |
| 9 | NOT (source OR destination) |
| 10 | NOT destination (inverse) |
| 11 | source OR (NOT destination) |
| 12 | NOT source (Inverse overwrite) |
| 13 | (NOT source) OR destination |
| 14 | NOT (source AND destination) |
| 15 | 1 (solid filled rectangle) |

Related Words:

PUT, GET

BLOAD

Command

Syntax: **BLOAD "filename", address**

Abbreviation : **BL**

Loads data or program segments into RAM from a Disk.

Parameters:

filename Name of the file as stored on the disk. The specifications of the hierarchical file system are permitted.

address This is a numerical expression for the first byte in RAM at which the information is to be stored. If an address is not specified, the address used during the BSAVE, which created the file, is used. If the file was not created by using BSAVE, an address must be specified.

Related Words:

BSAVE

BMOVE

Function

Syntax:**BMOVE source,destination,length****Abbreviation:****BM****Fast movement of contiguous memory blocks.****Parameters:****source** Integer expression for the first byte of the memory segment to be moved.**destination** Address to which the block of memory is to be moved.**length** Length (in bytes) of the block of memory to be moved.**Example:**

```
a$=SPACE$(32000)
PRINT AT(10,10);"Screen 1"
BMOVE XBIOS(3),VARPTR(a$),32000
CLS
PRINT AT(10,10);"Screen 2"
PAUSE 200
BMOVE VARPTR(a$),XBIOS(3),32000
```

This example moves the screen into the string variable, a\$, then clears the screen, and finally restores it by moving a\$ back onto the screen. The preferred method of doing this would be by using SGET and SPUT.

BOX**Function****Syntax:** **BOX** *x0,y0,x1,y1***Abbreviation:** **B**

This function draws an unfilled rectangular shape by specifying the diagonally opposed corners. The color of the rectangle is set by using the COLOR function.

Parameters:*x0* X coordinate of upper left corner of rectangle.*y0* Y coordinate of upper left corner of rectangle.*x1* X coordinate of lower right corner of rectangle.*y1* Y coordinate of lower right corner of rectangle.

The coordinates don't have to be on the screen area, but only those portions of the rectangle which appear on the screen will be displayed.

Example:

CLS

BOX 100,100,150,150

BOX 175,180,300,400

Related Words:

PBOX, PRBOX, RBOX

BPUT

Function

Syntax:**BPUT [#]channel_number,addr,bytes****Abbreviation:****BP**

This function saves an area of RAM to disk. Unlike BSAVE, the area saved need not be contiguous.

Parameters:

[#]channel_number An integer expression for the channel number associated with the disk drive as specified by the OPEN statement.

addr An integer expression for the address in memory where the first byte to be stored is located.

bytes An integer expression representing the number of bytes to be stored.

Example:

```
CLS
PRINT AT(15,1);"Top of screen"
PRINT AT(15,23);"Bottom of screen"
OPEN "O",#1,"SCRN.DAT"
BPUT #1,XBIOS(3)+16000,16000
BPUT #1,XBIOS(3),16000
CLOSE#1
PAUSE 200
CLS
BLOAD "SCRN.DAT",XBIOS(2)
PAUSE 200
```

This example swaps the top and bottom halves of the screen as the file is created, then loads the file from disk into screen memory and displays the result. Although this example is not terribly useful, it demonstrates how different data segments may be saved into one file.

Related Words:
BGET, BSAVE, BLOAD

BSAVE Command

Syntax: **BSAVE "filename",addr,len**

Abbreviation: **BS**

Saves data or program segments from RAM to disk.

Parameters:

filename Name of the file to be stored on the disk. The specifications of the hierarchical file system are permitted.

addr This is a numerical expression for the first byte in RAM of the information to be stored.

len Numeric expression specifying the length of the memory block to be saved.

Example:

BSAVE "PICTURE.PIC",XBIOS(2),32000

This example will BSAVE a file called "PICTURE.PIC" from screen memory to the default disk drive.

Related Words:
BLOAD, BPUT, BGET

C:
Function**Syntax:** **C:var(parameters)**

This function calls a machine language routine, with parameters passed to the called routine as in the C language. After the called routine is executed, a long word is returned in the D0 register.

Parameters:

var A numeric variable containing the starting address of the called routine.

parameters This is a list of the parameters to be passed to the called routine expressed either as 16-bit word, or 32-bit long words. Long words must be prefaced with 'L.'. Words need no preface, but the 'W.' preface may be used.

Example:

```
addr% = VARPTR(routine$)
a=C:addr%(15,L:0,W:-1)
```

This example only illustrates the calling procedure for clarification. It is not a functional program segment, as no executable code has been placed in memory.

Related Words:
BIOS, CALL, GEMDOS, XBIOS

CALL

Function

Syntax: **CALL** *variable* [(*parameters*)]

This function allows a machine language routine to be called. Parameter passing is permitted.

Parameters:

variable A numeric variable containing the start address of the machine language routine to be executed.

parameters This is an optional list of parameters to be passed to the machine language routine. The machine language routine receives the number of parameters as a 16-bit word, and the address of the 4-byte fields which contain each parameter as a 32-bit long word. Numbers are transferred as long words, and the starting address of a string is passed.

Example:**CALL** sample%(3,ctr%,a\$)

This is not a functional program segment, but only serves to illustrate the proper format.

Related Words:

BIOS, C:, GEMDOS, XBIOS

**CHAIN
Statement****Syntax:** **CHAIN "filespec"****Abbreviation:** **CH**

This function loads a program file into the work area of GFA BASIC and starts the program. This is useful when it is desirable to limit the area of memory being used to run a program, reserving the maximum area for graphics data, alternate screens, and other data. However, the value of variables in the original program is NOT preserved for the chained programs.

Parameters:

filespec This is the filename of the program to be loaded and executed. If no extension to the file name is given, the .BAS extension is assumed. Quotes surrounding the filename are not necessary.

Example:

Type in and save the two following example programs. Then run CHAIN1.BAS.

```
'CHAIN1.BAS
PRINT "This is program 1"
FOR ctr%=1 TO 10
    PRINT ctr%
NEXT ctr%
CHAIN "CHAIN2.BAS"
```

```
'CHAIN2.BAS
PRINT "This is program 2"
FOR ctr%=11 TO 20
    PRINT ctr%
NEXT ctr%
CHAIN "CHAIN.BAS"
```

CHAIN1.BAS is chained to CHAIN2.BAS. That is, after it runs through its loop of counting to 10, CHAIN2.BAS is loaded to count from 11 to 20. Then CHAIN2.BAS calls CHAIN1.BAS, and the procedure repeats. Press the <Control><Shift><Alternate> keys to stop.

CHDIR**Function****Syntax:** **CHDIR "directory"****Abbreviation:** **CHD**

This function changes the current directory, following the hierarchical file system, however, it is not possible to change drives with this function. Use CHDRIVE to change drives.

Parameters:

directory The name of the new directory to switch to. "\\" may be used to change to the root directory of the disk in the current drive.

Example:**CHDIR "PROGS"****Related Words:**

MKDIR, RMDIR

CHDRIVE**Function****Syntax:** **CHDRIVE *number***

This function sets the default disk drive.

Parameters:

n This is a numerical expression for the value of the drive to be set. This number may range from 1 to 15. (With 1 equal to Drive A.)

Example:

```
CHDRIVE 1  
FILES  
CHDRIVE 2  
FILES
```

This example will set the default drive first to disk A, then display all the files on that disk. Next the CHDRIVE function changes the drive to B, and displays all the files on that drive. (In the case of single drive systems, a TOS message will prompt the user to insert Disk B into Drive A.)

Related Words:

MKDIR, RMDIR

CHR\$

Function

Syntax: **CHR\$(value)**

Parameters:

value An integer numeric expression representing an ASCII code.

Example:

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

This example will print the letter "A", ASCII 65.

Related Words:

ASC

CIRCLE

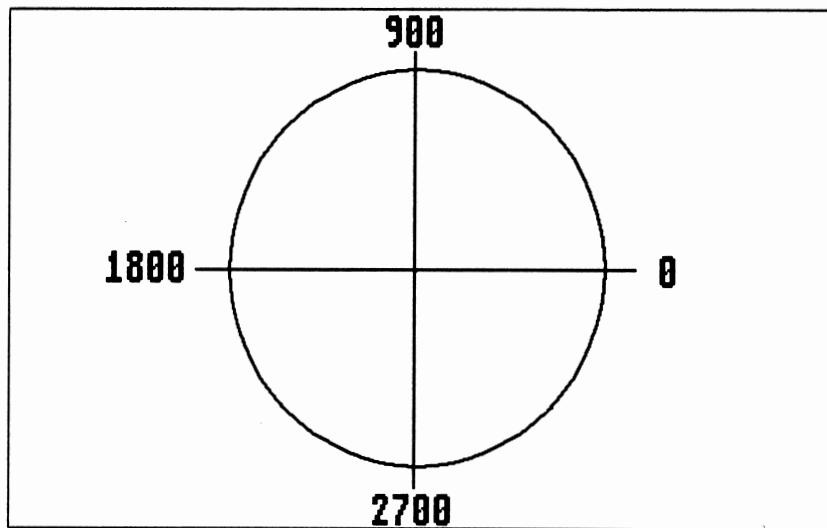
Function

Syntax: **CIRCLE x,y,r [arc1,arc2]****Abbreviation:** **C**

Draws a circle, or part of a circle (arc). Color for drawing must be selected with the COLOR statement.

Parameters:**x** X coordinate of center of circle or arc.**y** Y coordinate of center of circle or arc.**r** Radius (in pixels) of circle or arc.**arc1** Beginning angle of arc to be drawn. (Expressed in tenths of a degree (0-3600).**arc2** Ending angle of arc to be drawn. (expressed in tenths of a degree (0-3600).

0	Right
900	Top
1800	Left
2700	Bottom

Figure 2-2 CIRCLE**Example:**

```
CIRCLE 100,100,20  
CIRCLE 150,150,20,900,1300
```

This example draws a complete circle and an arc. The circle is centered at 100,100, with a radius of 20 pixels. The arc is centered at 150,150, with a radius of 20 pixels, and a starting angle of 90 degrees (at the top), and ending angle of 130 degrees.

Related Words:

PCIRCLE, ELLIPSE, PELLIPSE

CLEAR
Statement**Syntax:** **CLEAR****Abbreviation:** **CLE**

Clears all variables and arrays. Numeric variables become zero, and String variables become empty strings. Running a program automatically clears all variable.

Example:

```
a%=25
a$="This is a string"
PRINT a%
PRINT a$
CLEAR
PRINT a%
PRINT a$
```

CLEARW
Statement**Syntax:** **CLEARW** *number***Abbreviation:** **CLE W**

Clears the contents of the window selected.

Parameters:

number A numeric expression between one and four representing the number of the window (See OPENW).

Example:

```
OPENW 1
PRINT "Window 1"
PAUSE 100
CLEARW 1
```

Related Words:

OPENW, CLOSEW, FULLW

CLOSE

Statement

Syntax: **CLOSE [[#] number]**

Abbreviation: CL

Closes a data channel or closes a disk file.

Parameters:

number A numeric expression between 0 and 99 which states the number of the file as defined when the file was originally opened with the OPEN statement. This is an optional parameter, and if it is omitted, all open data channels will be closed.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT #1,"TEST"
CLOSE #1
OPEN "I",#1,"TEST.DAT"
INPUT #1,a$
CLOSE
PRINT a$
```

Related Words:

OPEN, PRINT #

CLOSEW**Statement**

Syntax: **CLOSEW** *number*

Abbreviation: **CL W**

Closes the window specified.

Parameters:

number A number or numeric expression which contains the window number.
CLOSEW 0 switches to a normal screen display. CLOSEW must be used in
any program when windows are opened or other programs may have
problems with screen graphics.

Example:

```
OPENW 1
OPENW 2
OPENW 3
OPENW 4
PAUSE 100
FOR ctr%=1 TO 4
  CLOSEW ctr%
  PAUSE 100
NEXT ctr%
```

Related Words:

OPENW, FULLW

CLR
Statement**Syntax:** **CLR variable [variable_list]**

Deletes variables from within a program.

Parameters:

variable The designator of the variable (not an array) to be erased.

Example:

```
a=1.999  
b%=2  
c$="This is a string"  
CLR a,b%,c$  
PRINT a  
PRINT b%  
PRINT c$
```

CLS
Statement**Syntax:** **CLS [#x]**

Clears the screen and places the cursor at the upper left 'Home' position.

Parameters:

#x When an optional number is specified, the number signifies a data channel opened by the OPEN statement.

Example:

```
PRINT "Watch closely!"  
FOR ctr%=0 to 99  
    PRINT "*";  
NEXT ctr%  
CLS  
PAUSE 100
```

COLOR
Statement

Syntax: *Color number*

Abbreviation: CO

Sets the color to be used by the drawing commands. (BOX, CIRCLE, DRAW, ELLIPSE, LINE, PLOT, and RBOX.) If GRAPHMODE 3 has been set, all graphics will be drawn in color 1, regardless of the color number specified. (Refer to Chapters 3 and 4 for examples.)

Parameters:

number Number of the preselected color register. Low resolution screens have a color range of zero to 15. Medium resolution screens have a color range of zero to three. High resolution screens may only use zero and one. In medium and low resolution modes, the color register may be set with the SETCOLOR statement.

Example:

```
FOR ctr%=0 TO 15  
    COLOR ctr%  
    CIRCLE 10+ctr%*4,10+ctr%*4,25  
NEXT ctr%
```

Related Words:

SETCOLOR

CONT

Command

Syntax: CONT**Abbreviation:** CON

This command may only be used in direct mode. It is used to continue execution of a program after it has been interrupted by the STOP command, or by pressing the <Control> <Shift> <Alternate> key combination. CONT will not work after executing a CLEAR, altering the program, or introducing new variables.

CTRL

System Variable

System variable containing the address of the VDI control block.

COS

Function

Syntax: COS(*number*)

Returns the trigonometric cosine of the value contained inside the parenthesis.

Parameters:

number This numeric expression contains the angle, expressed in radians, for which the cosine is to be calculated. To calculate degrees, multiply by PI/180.

Example:

```
PRINT COS(16)
PRINT COS(16)*PI/180
```

Related Words:

SIN, TAN

CRSCOL

Function

Syntax: $x\% = \text{CRSCOL}$

This function returns the present column position of the cursor.

Example:

```
CLS
cx\% = CRSCOL
cy\% = CRSLIN
PRINT AT(10,10); "TEST"
PRINT AY(cx\%, cy\%);
```

In this example, the location of the cursor is stored in cx% and cy%. After printing the string "TEST" at screen position 10,10, the cursor is returned to its original position.

Related Words:

CRSLIN

CRSLIN

Function

Syntax: $x\% = \text{CRSLIN}$

This function returns the present column position of the cursor.

Example:

```
CLS  
cx%=CRSCOL  
cy%=CRSLIN  
PRINT AT(10,10);"TEST"  
PRINT AY(cx%,cy%);
```

In this example, the location of the cursor is stored in cx% and cy%. After printing the string "TEST" at screen position 10,10, the cursor is returned to its original position.

Related Words:

CRSCOL

CVD

Function

Syntax: **CVD (a\$)**

Changes an 8-byte character string in Microsoft BASIC compatible format into a number.

Parameters:

a\$ Any character string expression. Only the first 8 bytes of the string, if the string is longer than 8 bytes, are changed.

Example:

```
a$="12345678"  
b$="This is a string"  
PRINT CVD(a$)  
PRINT CVD(b$)
```

Related Words:

CVF, CVI, CVL, CVS, MKI\$, MKL\$, MKS\$, MKF\$, MKD\$

CVF**Function****Syntax:** **CVF(a\$)**

Changes a 6-byte character string in GFA BASIC compatible format into a number.

Parameters:

a\$ Any character string expression. Only the first 6 bytes of the string, if the string is longer than 6 bytes, are changed.

Example:

```
a$="123456"  
b$="This is a string"  
PRINT CVF(a$)  
PRINT CVF(b$)
```

Related Words:

CVD, CVI, CVL, CVS, MKI\$, MKL\$, MKS\$, MKF\$, MKD\$

CVI**Function****Syntax:** **CVI(a\$)**

Changes a 2-byte character string into a 16-bit integer number.

Parameters:

a\$ Any character string expression. Only the first 2 bytes of the string, if the string is longer than 2 bytes, are changed.

Example:

```
a$="123456"  
b$="This is a string"  
PRINT CVF(a$)  
PRINT CVF(b$)
```

Related Words:

CVD, CVF, CVL, CVS, MKI\$, MKL\$, MKS\$, MKF\$, MKD\$

CVL**Function**

Syntax: **CVL (a\$)**

Changes a 4-byte character string into a 32-bit integer number.

Parameters:

a\$ Any character string expression. Only the first 4 bytes of the string, if the string is longer than 4 bytes, are changed.

Example:

```
a$="1234"  
b$="This is a string"  
PRINT CVL(a$)  
PRINT CVL(b$)
```

Related Words:

CVD, CVF, CVI, CVS, MKI\$, MKL\$, MKS\$, MKF\$, MKD\$

CVS**Function**

Syntax: **CVS (a\$)**

Changes a 4-byte character string in Atari BASIC compatible format into a 32-bit number.

Parameters:

a\$ Any character string expression. Only the first 4 bytes of the string, if the string is longer than 4 bytes, are changed.

Related Words:

CVD CVF, CVI, CVL, MKI\$, MKL\$, MKS\$, MKF\$, MKD\$

DATA**Statement**

Syntax: **DATA constant, constant, constant....**

Holds the data which can be used by the program with the READ statement.

Parameters:

constant A boolean, numeric, or string constant. Each constant must be separated, by a comma. Numeric constants may be in decimal, hexadecimal, octal, or binary format. Phrases which contain a comma may be placed within quotes.

Example:

```
READ a,b,c,d,e,a$  
PRINT a  
PRINT b  
PRINT c  
PRINT d  
PRINT e  
PRINT a$  
DATA 1,2,3,4,5,Test
```

Related Words:

READ, RESTORE

DATE\$

Function

Syntax:

PRINT DATE\$

Creates a character string which contains the systems date in the format:

MM/DD/YYYY

Example:

PRINT DATE\$

Related Words:

TIME\$

DEC

Function

Syntax: **DEC** *variable*

This is an arithmetic function which subtracts one from a given variable. The advantages of this function are that it saves storage space, and executes more quickly than the standard *a=a-1* form.

Parameters:

variable Any numeric expression.

Example:

```
t=TIMER  
FOR ctr%=1 TO 10000  
    DEC a%  
NEXT ctr%  
PRINT (TIMER-t)/200  
a%=0  
t=TIMER  
FOR ctr%=1 TO 10000  
    a%=a%-1  
NEXT ctr%  
PRINT (TIMER-t)/200
```

Related Words:

INC

DEFFILL**Function****Syntax:****DEFFILL color, fill_1, fill_2**

or

DEFFILL color, variable\$**Abbreviation****DEFF**

This function defines the fill color and pattern for shapes drawn with FILL, PBOX, PCIRCLE, PELLIPSE, and PRBOX. User-defined fill patterns may also be used. Parameters not specified will be left unchanged.

Parameters:

color The palette for the fill pattern. Zero or one for high resolution (monochrome monitors), zero through three for medium resolution, and zero through 15 for low resolution. The palettes may be set to any shade by using SETCOLOR. If GRAPHMODE 3 (XOR mode) has been set, the pattern fill will always be in color 1 (usually black), regardless of the color selected.

fill_1 Type of fill pattern

- 0 Empty
- 1 Solid fill
- 2 Dots
- 3 Straight lines
- 4 User-defined

fill_2 Selection of 24 different dot patterns, or 12 line patterns, as shown in Figure 2-3.

- a\$ A user-defined fill pattern. a\$ is defined as a 16 by 16 bit pattern array. The bit pattern is formed by linking 16 bit pattern numbers with the function MKI\$, with each line representing one line of the array. If the bit is set, a dot shows in the pattern. For example,

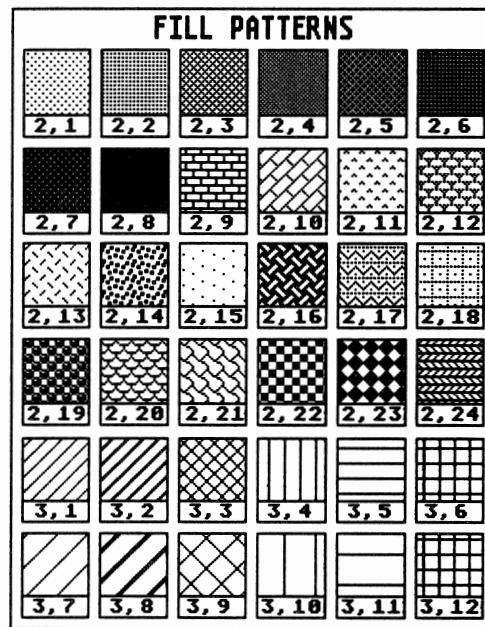
```
FOR ctr%=1 TO 16  
a$=a$+MKI$(65535)  
NEXT ctr%
```

This fill pattern will produce a solid block. 65535 in binary format is the bit pattern:

```
1111111111111111
```

Example:

```
BOX 100,100,400,300  
DEFFILL 1,2,9  
FILL 50,100  
DEFFILL , ,10  
FILL 320,100
```

Figure 2-3 Fill Patterns

DEFFN
Statement

Syntax: **DEFFN** *function_name*, [*variable_list*] = *expression*

Permits the user to define a specific function.

Parameters:

function_name Any name to identify the function.

variable_list An optional parameter consisting of numeric or string variables, to be passed to the function. Each element in this list must be separated by a comma.

expression Any program statement providing a numeric or character string, matching the type of the expression.

Example:

```
DO
  INPUT x
  PRINT x, FN three,FN MULT(10)
LOOP
'
DEFFN MULT(y)=y*FN(Three)
DEFFN Three=3*x
```

DEFLINE

Function

Syntax:**DEFLINE** *line_style, line_width, begin, end***Abbreviation :**

DE

Sets the line style, line width, type of line, and the beginning and ending points to be used with **BOX**, **CIRCLE**, **ELLIPSE**, **LINE** and **RBOX**.

Parameters:*line_style* Line style; dotted or solid. Refer to Figure 2-4 for details.*line width* Numeric expression representing the number of pixels for line width.*begin* Type of beginning point for line.

- 0 Normal
- 1 Arrow
- 2 Rounded

end Type of end point for line.

- 0 Normal
- 1 Arrow
- 2 Rounded

Example:

```
DEFLINE 2,4,1,2  
LINE 10,100,40,150
```

Figure 2-4 Line Styles

S = 1 ——————

S = 2 -----

S = 3

S = 4

S = 5 - - - - -

S = 6 - - - - -

DEFLIST Command

Syntax: **DEFLIST 0**

DEFLIST 1

Abbreviation: **DEFLIS**

Defines the listing format. For DEFLIST 0, commands and function names are displayed in capitol letters (in the Editor or in a printed listing). All variables are listed in lowercase letters. This is the convention chosen for clarity in presenting the program listings in this book.

DEFLIST 1 displays the listing with the initial letter in upper case for all command, function, and variable names.

DEFMARK**Function****Syntax:** **DEFMARK** *color, type, size***Abbreviation:** **DEFM**

Defines the color, type, and size of the corner points to be marked by POLYMARK. If GRAPHMODE 3 (XOR mode) has been selected, all marks are drawn in color 1, regardless of the value specified.

Parameters:

color Pre-selected color. Values are zero or one for high resolution, zero to three for medium resolution, or zero to fifteen for low resolution.

type Type of mark:

- 1 Dot
- 2 Plus (+) sign
- 3 Asterisk (*)
- 4 Rectangle
- 5 Cross
- 6 Hash

size Size of the mark. Values 0, 20, 40, 60, 80, or 100 are suitable values for the size.

DEFMOUSE**Function****Syntax:****DEFMOUSE** *number*

or

DEFMOUSE *variable\$***Abbreviation:****DEFMO**

Selects one of the predefined mouse forms, or allows the user to define a custom mouse form.

Parameters:

number A numeric expression representing an integer number between zero and seven, representing one of the predefined mouse forms

Predefined Mouse Forms

- 0 Arrow
- 1 Expanded X
- 2 Bee
- 3 Pointing Hand
- 4 Open Hand
- 5 Thin Crosshairs
- 6 Thick Crosshairs
- 7 Bordered Crosshair

variable\$ A user defined mouse form consisting of a 16 pixel by 16 pixel shape. The mask, used to create a border, and the actual mouse cursor must be defined. One of the pixels must also be defined as the starting point, or "hot" spot which all mouse functions, such as MOUSEX will refer to.

The string variable is composed as follows:

a\$=MKI\$(x)	Starting point x coordinate
a\$=a\$+MKI\$(y)	Starting point y coordinate
a\$=a\$+MKI\$(1)	
a\$=a\$+MKI\$(0)	Mask Color, usually zero
a\$=a\$+MKI\$(1)	Cursor color, usually one
a\$=a\$+MKI\$(bit_pattern)	Bit-pattern of user-defined shape

Example:

```
DEFMOUSE 0
@mouse_click
DEFMOUSE 1
@mouse_click
DEFMOUSE 2
@mouse_click
DEFMOUSE 3
@mouse_click
DEFMOUSE 4
@mouse_click
DEFMOUSE 5
@mouse_click
DEFMOUSE 6
@mouse_click
DEFMOUSE 7
@mouse_click

PROCEDURE mouse_click
REPEAT
    x=MOUSEK
UNTIL x=1
RETURN
```

This example redefines the mouse shape to one of the eight predefined mouse forms every time the left mouse button is pressed.

DEFNUM**Function****Syntax:** DEFNUM *variable***Abbreviation:** DEFN

This function rounds all numbers to a specified number of digits, between 3 and 11, before output to a device. This corrects any rounding errors which may have occurred.

Example:

```
DIM a%(10)
DEFNUM 5
FOR ctr%=0 TO 10
  a% (ctr%)=RND*10
  PRINT a% (ctr%)
NEXT ctr%
```

In order to illustrate the effect of DEFNUM, run the example program twice. Once as listed, and a second time with the second line, DEFNUM 5, removed.

Related Words:**PRINT USING****DEFTEXT****Function****Syntax:** DEFTEXT *color, style, rotation, size***Abbreviation:** DEFT

This function defines the color, style, rotation, and size of the text to be displayed by executing the TEXT statement. If GRAPHMODE 3 (XOR mode) has been set, the text will be displayed in color one, regardless of the value specified.

Parameters:

- color** A numeric value representing the color register previously set by using the SETCOLOR function. High resolution (monochrome) screens may only be set to zero or one. Medium resolution (color) screens may be set to a value between zero and three, while low resolution (color) screens may be set to any value between zero and fifteen.
- style** This numeric expression defines the style to be used when the text is displayed.

<u>Value</u>	<u>Text Style</u>
--------------	-------------------

0	Normal
1	Bold
2	Ghost (light)
4	Italic
8	Underlined
16	Outlined

Style is a bit-wise value, and the values may be combined to produce a desired effect. For instance, the value 17 will produce characters which are a combination of bold(1) and outline (16).

- rotation** Defines the rotation (if any) of the text in tenths of a degree. However, only the following angles of rotation are possible:

0	0 degrees
900	90 degrees
1800	180 degrees
2700	270 degrees

- size** This value defines the size of the text characters in graphic dots (pixels):

4	Icon (very small)
6	Normal (color) 8*8 pixels
13	Normal (mono)
32	Enlarged

DFREE
Function**Syntax:** **DFREE(*n*)**

Returns the amount of free storage space remaining on a disk.

Parameters:

n A numeric expression for the value number of the drive (between 0 and 15) to be checked.

Example:

```
PRINT DFREE(0)
```

DIM
Command**Syntax:** **DIM variable(*index*)****Abbreviation:** DI**Parameters:**

Provides a means for storing data within a program.

variable Any numeric, alphanumeric, or boolean array variable name.

index Number of indexes to a variable. Smallest permitted value is zero, largest is 65535 for multi-dimensional arrays. One dimensional arrays are limited only by the available memory.

Example:

```
DIM a%(10)
DEFNUM 5
FOR ctr% = 0 TO 10
    a%(ctr%) = RND * 10
    PRINT a%(ctr%)
NEXT ctr%
```

Related Words:

ERASE

DIM?

Function

Syntax: DIM?(*field()*)

Determines the number of elements in an array.

Parameters:

field() Any array variable.

Example:

```
DIM a$(10), b$(2,3,4)
PRINT DIM?(a$())
PRINT DIM? (b$())
```

Related Words:

DIM

DIR**Function**

Syntax: **DIR(*filespec*) [TO *file*]**

Lists the files on a disk. This function does not list files within folders of the current directory. The hierarchical file system may be used.

Parameters:

filespec Any file name. The file name is specified by placing the letter of the drive in front of the file name, followed by a colon. If question marks are present in the name, they are treated as wild cards, and all files matching the name, except for the question marks, will be listed.

If an asterisk (*) is placed in the file name, it means to disregard the entire or remaining part of the filename.

If no arguments are used, DIR will produce a list of all the files on the current drive in the current folder.

The hierarchical file system is also permitted as part of the file specification.

file The output produced by DIR may be redirected to an alternate device by specifying the device in the TO portion of the function.

Redirection may be to:

PRN:	Printer
LST:	Printer
AUX:	Disk Drive
MIDI:	Midi Port
VID:	Screen

Example:

DIR	Returns all files in the current drive and folder.
DIR "B:"	Returns all files on Drive B.
DIR "A:*.PRG"	Returns all .PRG files in the current directory of Drive A.
DIR "A:HA???.PRG"	Returns all files in the current directory of Drive A, which begin with the characters "HA", are five characters in length, and end with the .PRG extension to the file name.
DIR "A:.*" TO ".PRN"	Lists all files on disk A, and redirects the output to an attached printer.

DIR\$**Function****Syntax:** **DIR\$(number)**

This function returns the name of the active directory for the disk drive represented by *number*. If the active directory is the root directory, an empty string is returned.

Parameters:

number A numeric expression representing the number of the disk drive. Drive A = 1, B = 2, etc. 0 represents the currently selected drive.

Example:

```
MKDIR "SAMPLE"
CHDIR "SAMPLE"
PRINT DIR$(0)
```

DIV
Function**Syntax:** **DIV** *variable, number*

Provides a fast division function, divides *variable* by *number*. This is the same as using the '/' symbol, the chief benefit is increased speed of execution.

Parameters:

variable Either a numeric variable or numeric array variable.

number Any numeric expression.

Example:

```
DIM a%(10000)
FOR ctr%=1 TO 10000
  a% (ctr%)=5
NEXT ctr%
t=TIMER
FOR ctr%=1 TO 10000
  DIV a% (ctr%),5
NEXT ctr%
PRINT "USING DIV ";(TIMER-t)/200;" Secs."
t=TIMER
FOR ctr%=1 TO 10000
  a% (ctr%)=a% (ctr%)/5
NEXT ctr%
PRINT "USING / ";(TIMER-t)/200;"Secs."
```

DO...LOOP
Statement**Syntax:**

```
DO  
    Program statement  
    Program statement  
LOOP
```

Abbreviation:

```
DO  
L
```

This statement creates an endless loop, which can only be left by using an EXIT statement, or by stopping the program by pressing the <Control>, <Shift>, and <Alternate> keys simultaneously. This may be used to create a control structure which checks for branches to procedures within a program.

Example:

```
DO  
    EXIT IF inp(2)=13  
    INC ctr%  
    PRINT ctr%  
LOOP  
PRINT "Out of loop"
```

DPEEK
Function**Syntax:**

DPEEK *address*

DPEEK examines the contents of a two consecutive byte area of memory. The memory address must be an even number. The value in this area of memory is returned as a two byte 16-bit value.

Parameters:

address A numeric expression specifying an address in the main area of memory.

Example:

```
a$="A"  
desc=ARRPTR(a$)  
addr=LPEEK(d)  
length=DPEEK(d+4)  
ascii=PEEK(a)  
PRINT "Descriptor Address: ";desc  
PRINT "Address          : ";addr  
PRINT "Length           : ";length  
PRINT "ASCII Value     : ";ascii
```

Related Words:

PEEK, LPEEK

DPOKE

Function

Syntax: DPOKE *address,number*

Abbreviation: DP

Stores 2 bytes into an area of main memory at a specified address.

Parameters:

address A numeric expression which specifies an even address in memory.

number A numeric expression between 0 and 65535.

Example:

```
a$="A"  
ctr%=ARRPTR(a$)  
DPOKE ctr%+4,4  
z=VARPTR(a$)  
LPOKE z,111638594  
PRINT a$  
POKE z,67  
PRINT a$
```

Related Words:

POKE, LPOKE, SPOKE, SLPOKE

DRAW
Function**Syntax:**

**DRAW [TO] x1,y0 [TO x1,y1]
DRAW x0,y0 TO x1,y1 [TO x2,y2]**

Abbreviation:**DR**

Draws points and connects two or more points with straight lines.

Parameters:

x0,y0 x and y coordinates of first point.

x1,y1 x and y coordinates of second point.

x2,y2 x and y coordinates of third point.

Example:

DRAW 10,10 ! Marks first point

DRAW TO 200,10 ! Draws straight line between points

DRAW 200,10 TO 200,190 TO 10,190 TO 10,10

EDIT
Statement**Syntax:****EDIT****Abbreviation:****ED**

Exits program and returns to the editor, without the alert box at the end of the program.

Example:

PRINT "This is the program"

PRINT "Back to the Editor without Alert Box"

EDIT

ELLIPSE**Function**

Syntax: **ELLIPSE x,y,r [,arc1,arc2]**

Draws an ellipse, or part of a ellipse (arc).

Parameters:

x X coordinate of center of ellipse or arc.

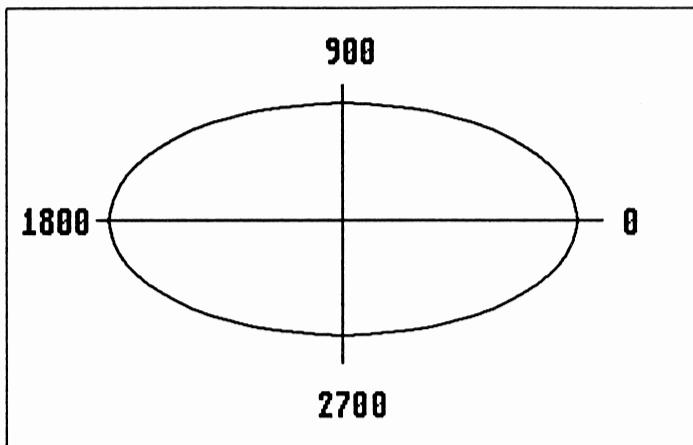
y Y coordinate of center of ellipse or arc.

r Radius (in pixels) of ellipse or arc.

arc1 Beginning angle of arc to be drawn. (In tenths of a degree 0-3600).

arc2 Ending angle of arc to be drawn. (In tenths of a degree 0-3600).

0	Right
900	Top
1800	Left
2700	Bottom

Figure 2-5 Ellipse**Example:**

```
ELLIPSE 100,100,20  
ELLIPSE 150,150,20,900,1300
```

This example draws a complete ellipse, centered at 100,100, with a radius of 20, and an arc centered at 150,150, with a radius of 20, and a starting angle of 90 degrees (at the top) ending at 130 degrees.

Related Words:
PELLIPSE

END
Statement

Syntax: **END**

Closes all open files and terminates the program. The END statement may be used at any part of the program. Its use at the physical end of the program is optional.

Example:

```
DO  
    INC ctr%  
    IF ctr% = 10000  
        END  
    ENDIF  
LOOP
```

EOF
Function

Syntax: **EOF([#]number)**

This function determines whether the file pointer for the open file assigned the channel number specified by *number* has reached the end of the file. If the pointer has reached the end of the file, -1 (TRUE) is returned, otherwise, the value returned is 0 (FALSE). This function is only used with disk files.

Parameters:

number This is an integer expression between 0 and 99 referring to the channel number for a file opened by the OPEN statement.

Example:

```
OPEN "O",#1,"TEST.DAT"  
PRINT#1,"TEST DATA"  
CLOSE#1  
OPEN "I",#1,"TEST.DAT"  
DO  
    PRINT INP(#1),EOF(#1)  
    EXIT IF EOF(#1)  
LOOP
```

EQV
Operator

Syntax: **condition EQV condition**

This is a logical mathematical operator for logically comparing the equivalency of conditions.

Condition	EQV	Condition	Result
-1	EQV	-1	-1
-1	EQV	0	0
0	EQV	-1	0
0	EQV	0	-1

"If Condition 1 is true AND condition 2 is true then they are equivalent."

Example:

```
a$="A"
b$="B"
IF a$="A" EQV b$="B"
    PRINT "EQV satisfied"
ENDIF
IF a$="B" EQV b$="A"
    PRINT "EQV satisfied"
ENDIF
```

ERASE Statement

Syntax: **ERASE array()**

Abbreviation: **ER**

Erases an array from memory and releases the previously dimensioned memory area, allowing reserved memory to be used for other purposes.

Parameters:

array() An array of variables.

Example:

```
DIM a(6000)
PRINT FRE(0)
ERASE a()
PRINT FRE(0)
```

ERR

System Variable

Syntax: **ERR**

Returns the error code for any error that has occurred, as shown in Table 2-1.

Table 2-1
GFA BASIC Error Messages

<u>Error Number</u>	<u>Meaning</u>
0	Division by Zero
1	Overflow
2	Number not integer (-21473648 to +2147483647)
3	Number not byte (0 to 255)
4	Number not word (0 to 65535)
5	Square root only for positive numbers
6	Logarithm only for numbers greater than zero
7	Undefined error
8	Memory full
9	Function or command not possible
10	String too long, maximum size is 32767 characters
11	Not GFA BASIC Vx.x program
12	Program too long, memory full
13	Not GFA BASIC program file
14	Field dimensioned twice
15	Field not dimensioned
16	Field index too large
17	Dim index too large
18	Wrong number of indexes
19	Procedure not found
20	Label not found
21	On OPEN only "I"nput, "O"utput, "R"andom, "A"ppend, "U"pdate allowed
22	File already opened
23	File number wrong
24	File not opened
25	Input wrong, not numeric
26	End of file reached
27	Too many points for Polyline/Polyfill, maximum 128 points.
28	Field must be one dimensional
29	Number of points larger than field
30	Merge not an ASCII file
31	Merge line too long, Merge aborted
32	Syntax Error Program aborted
33	Label not defined
34	Insufficient data
35	Data not numeric

36	Syntax error in data, unpaired quotes
37	Disk full
38	Command not possible in direct mode
39	Program error, GOSUB not possible
40	Clear not possible in FORNEXT loops and procedures
41	CONT not possible
42	Too few parameters
43	Expression too complex
44	Function not defined
45	Too many parameters
46	Parameter wrong, must be numeric
47	Parameter wrong, must be string
48	OPEN "R" Record length wrong
50	Not an "R" file
51	Only one field per open "R" allowed
52	Fields larger than record length
53	Too many fields, maximum of 9 permitted
54	GET/PUT fields, string length wrong
55	GET/PUT record number wrong
56	String has wrong length in SPRITE
90	Error in LOCAL
91	Error in FOR
92	RESUME (NEXT) not possible. FATAL, FOR or LOCAL
100	Version number and copyright for GFA BASIC

Example:

```
ON ERROR GOSUB err_routine
PRINT SQR(1)

PROCEDURE err_routine
  PRINT "Error number: ";ERR
  RESUME NEXT
RETURN
```

Related Words:
FATAL, ERROR, ON ERROR

ERROR
Statement**Syntax:** **ERROR** *number***Abbreviation:** **ERR**

Simulates the occurrence of the error with the error code number specified.

Parameters:*number* An integer expression between -128 and 127.**Example:**

```
PRINT "Enter an error number:";  
INPUT e  
ERROR e
```

Enter '100' to check your version number of GFA BASIC.

EVEN
Function**Syntax:** **EVEN**(*number*)

Determines if a number is even (2,4,6,8, etc.). A value of -1 (TRUE) is returned if the number is even.

Parameters:*number* A numeric expression. If the variable is an even number, a -1 (TRUE) is returned. If the number is odd, then a value of zero (FALSE) is returned.**Example:**

```
FOR ctr%=1 to 100  
  IF EVEN(ctr%)  
    PRINT ctr%  
  ENDIF  
NEXT ctr%
```

Related Words:
ODD

EXEC

Function

Syntax:**EXEC** *flag, name, cmd, env*

Loads and executes a machine language or compiled program from disk.

Parameters:

flag A numeric expression:
0 Load and run
3 Load only

name The name of the program file to load. The hierarchical file system is permitted.

cmd Command line. When this is not an empty string, and C\$ holds the command, *cmd* is defined as:

CHR\$(len(c\$))+c\$+CHR\$(0)

env Environment string (usually empty)

Example:

```
RESERVE FRE(0)10000
EXEC 0,"TEST.PRG","",""
RESERVE FRE(0)+10000
```

Related Words:

RESERVE, HIMEM

EXIST

Function

Syntax:**EXIST**(*file_spec*)

Determines whether or not a particular file is present on the disk. If the file is present -1 (TRUE) is returned. If the file is not found, a zero (FALSE) is returned. TRUE will be returned only if a filename exactly matches the name used in calling the function. Wildcards (?) and (*) are permitted.

Parameters:

file_spec A file name, which follows the hierarchical file system.

Example:

```
OPEN "O",#1,"TEST.DAT"!Create a test file
CLOSE#1
PRINT EXIST("FILE")    !Returns 0, File not found
PRINT EXIST("TEST.*")   !Returns 1, Match found
PRINT EXIST("*.*)"      !Returns 1, Match found
```

Related Words:

DIR

EXIT IF
Statement

Syntax: **EXIT IF** *condition*

Abbreviation: E

Enables an exit from a DO...LOOP, FOR...NEXT, REPEAT...UNTIL, or WHILE...WEND loop when the conditions specified are satisfied.

Parameters:

condition Any conditional statement.

Example:

```
CLS
PRINT "Press any key Press <Return> to Exit"
DO
  a%=inp(2)
  EXIT IF a%=13
  IF a%<>0
    PRINT "You pressed ";CHR$(a%)
  ENDIF
LOOP
```

EXP
Function**Syntax:** EXP(*number*)

Calculates the value of an exponent. Actually calculates the value of 2.718281828 raised to the power specified by *number*. (2.718281828^{*number*})

Parameters:

number Any numeric expression.

Example:

```
e=2.718281828
PRINT EXP(100)
PRINT EXP(4)
PRINT e^4
```

FALSE
System Constant**Syntax:** FALSE

This is simply another way to express the value of a condition when the result is zero (FALSE). Aids in making the program code segment more readable.

Example:

```
IF "A"="B"
  flag!=TRUE
ELSE
  flag!=FALSE
ENDIF
PRINT flag!
```

Naturally, since A does not equal B, the value for the boolean variable *flag!* is set as FALSE, and a zero is printed.

Related Words:
TRUE

FATAL**System Variable****Syntax:****FATAL**

Normal error conditions return a value of zero. If the address of the last executed command is no longer known, as is the case with errors that lead to the display of "bombs", the value -1 is returned.

Example:

```
ON ERROR GOSUB err_routine
ERROR 5
'
PROCEDURE err_routine
  PRINT "Value of FATAL=";FATAL
RETURN
```

Try this example substituting different values in the ERROR statement (second line).

FIELD**Function****Syntax:****FIELD[#] number, field_len AS string_variable**

This function divide records into arrays. The expression *AS string_variable* portion may be repeated as needed, by separating with commas.

Parameters:

[#]number An integer expression between 0 and 99 which refers to the data channel assigned by the OPEN statement.

field_len An integer expression which defines the length of the field.

string_variable A string variable (not an array) which accepts the data field. The length of this string may not be changed if it is required by a GET or PUT statement. Any changes should be made using the LSET or RSET functions.

Example:

```
OPEN "R",#1,"TEST.FIL",50
FIELD #1,20 AS A$,30 AS B$
FOR ctr%=1 TO 4
    INPUT "Name #1, Name #2:",v$,n$
    LSET a$=v$
    LSET b$=n$
    PUT#1,ctr%
NEXT ctr%
FOR ctr%=4 DOWNTO 1
    GET#1,ctr%
    PRINT a$,b$
NEXT ctr%
CLOSE#1
```

FILES**Function****Syntax:**

FILES *file_spec* [TO *file*]

Abbreviation:

FILE

Lists the files and folders in the current directory on the selected drive.

Parameters:

file_spec A file name which follows the same specifications as the DIR function.
May use the hierarchical file system.

file Permits redirection of the list produced by this function to devices other than the default screen. Adding the statement TO "LST:" will send the files to an attached printer, while TO "A:FILES.TXT" will create a disk file named "FILES.TXT" containing the information.

Example:

FILES "A:.*"	!Displays contents of Disk A
FILES "A:*.PRG"	!Displays all .PRG files on Disk A
FILES "A:.*" TO ":LST"	!Sends contents to printer
FILES "A:.*" TO "A:FILES.TXT"	!Sends contents to disk

Related Words:

DIR

FILESELECT

Function

Syntax: FILESELECT *filespec, filename, select\$***Abbreviation:** FILESE

Creates the standard GEM file selector box on the screen.

Parameters:**filespec** This is a search path allowing the use of the hierarchical file system. This string variable must contain at least "*.*", in which case all files in the root directory of the current disk are shown in the fileselector box.**filename** This optional variable contains the name of the default file which will appear under "Selection:" in the file selector box. Normally, this is any empty or "Null" string; "".**select\$** Any string variable which will accept the name of the file selected by the user. When CANCEL is selected, this will be an empty string (Null string).**Example:**

```
DO
  FILESELECT "\*.*",b$,a$
  EXIT IF a$=""
  PRINT "You Selected "a$
LOOP
```

FILL

Function

Syntax: FILL *x,y***Abbreviation:** FI

Fills a previously drawn, bordered area with a preselected fill pattern. (See DEFFILL.)

Parameters:

x X coordinate from which filling process will begin.

y Y coordinate from which filling process will begin.

The origin of the screen or window is always the top left corner (0,0).

Example:

```
BOX 10,10,100,100  
BOX 5,5, 105,105  
DEFFILL 1,2,9  
FILL 50,50  
DEFFILL 2,2,9  
FILL 7,7
```

Related Words:

DEFFILL

FIX

Function

Syntax:

FIX(variable)

Returns the integer portion of a numeric expression. If the expression is a positive number, the effect is the same as the INT statement. FIX complements FRAC. FIX is identical to TRUNC.

Parameters:

variable Any numeric expression. Note that FIX(1.99) returns a value of 1.

Example:

```
a=PI  
PRINT a  
PRINT FIX(a)  
PRINT FIX(12)  
PRINT FIX(1.99)
```

Related Words:

TRUNC, INT

FOR...NEXT
Statement**Syntax:**

```
FOR var=a [DOWN]TO b [STEP c]
    program statement
NEXT var
```

Abbreviation: F [DOWN]TO STEP

Creates a loop which is executed the number of times specified by the FOR statement which begins the loop.

Parameters:

var Numeric variable incremented by the FOR statement.

a Initial value assigned to *var*.

b Final value assigned to *var*.

c An optional value by which *var* may be incremented each time the loop is repeated. If this optional variable is omitted, *var* is incremented by one each time.

If the optional statement, DOWNTO, is used instead of TO, then *var* is decremented instead of incremented.

Using an integer variable (%) in a loop increases the speed of execution.

Example:

```
FOR ctr%=1 TO 100
    PRINT ctr%
NEXT ctr%
FOR ctr%=1 TO 100 STEP 5
    PRINT ctr%
NEXT ctr%
FOR ctr%=100 DOWNTO 1
    PRINT ctr%
NEXT ctr%
```

Related Words:

WHILE...WEND, REPEAT...UNTIL

FORM INPUT

Statement

Syntax:**FORM INPUT *max_len,string_variable*****Abbreviation:****F MINPUT**

Enables the input of a character string during program execution, but limits the input to a preset length. When the maximum set length is reached, the bell sounds. Both the <Delete> and <Backspace> keys may be used for normal editing corrections. The left and right cursor keys may be used to move the cursor to a desired character in the string, and the up and down cursor keys may be used to move the cursor to the beginning and end of the text.

FORM INPUT works only in the Insert mode.

Special characters can be entered in three different ways:

- By pressing the <Alternate> key at the same time as another key.
- By pressing the <Control> and <S> key simultaneously, followed by another key.
- By pressing the <Control> and <A> keys simultaneously, followed by the ASCII codes of the chosen character. (If the ASCII code is less than 26, you must then press any other non-numeric key.)

Parameters:

max_len An integer expression representing the maximum length of the character string.

string_variable The identifier of the string variable.

Example:

```
PRINT "Enter your name"
PRINT "(max. 15 letters)"
PRINT AT(1,15);
FORM INPUT 15,name$
PRINT "Your name is ",name$
```

Related Words:

INPUT, LINE INPUT, FORM INPUT AS

FORM INPUT AS

Statement

Syntax: **FORM INPUT** *max_len AS string_variable***Abbreviation:** **F MINPUT**

Permits a character string to be changed during program execution, but limits the input to a preset length. This statement works the same way as FORM INPUT, except that the contents of the old string are printed on the screen to be edited.

When the maximum set length is reached, the bell sounds. Both the <Delete> and <Backspace> keys may be used for normal corrections. The left and right cursor keys may be used to position the cursor at a desired character in the string. The up and down cursor keys may also be used to position the cursor.

FORM INPUT AS works only in the insert mode.

Special characters can be entered in three different ways:

- By pressing the <Alternate> key at the same time as another key.
- By pressing the <Control> and <S> key simultaneously, followed by another key.
- By pressing the <Control> and <A> keys simultaneously, followed by the ASCII codes of the chosen character. (If the ASCII code is less than 26, you must then press any other non-numeric key.)

Parameters:

max_len An integer expression representing the maximum length of the character string.

string_variable The identifier of the string variable.

Example:

```
name$="John Smith"
PRINT "Enter your name"
PRINT "(max. 15 letters)"
PRINT AT(1,15);
FORM INPUT 15 as name$
PRINT "Your name is ";name$
```

Related Words:

INPUT, LINE INPUT, FORM INPUT

FRAC
Function**Syntax:** **FRAC(*number*)**

Returns the digits after the decimal point in a real number. This function is not very accurate, and should not be used where accuracy is desired.
FRAC complements TRUNC.

Parameters:*number* Any integer expression.**Example:**

```
a=9.34567  
b=111.12345  
PRINT FRAC(a)  
PRINT FRAC(b)  
PRINT FRAC(3.1415)  
PRINT TRUNC(a)+FRAC(a)
```

Related Words:
INT, TRUNC

FRE
Function**Syntax:** **FRE(*numeric_expression*)**

Calculates the amount of free storage space in main memory in bytes. All unused bytes in memory are searched for and collected.
This process is often referred to as "Garbage Collection".

Parameters:*numeric_expression* Any numeric expression. This value is disregarded.

Example:

```
PRINT FRE(0)
DIM a$(1000)
PRINT FRE(0)
```

Related Words:

DIM, RESERVE, ERASE

FULLW

Statement

Syntax:

FULLW *number*

Enlarges the designated window to the full screen size. If the designated window has not been opened, this statement will open the window.

Parameters:

number A numeric expression between 1 and 4 representing the window number.

Example:

```
OPENW 1
PAUSE 100
FULLW 1
```

Related Words:

OPENW, CLOSEW

GB

System Variable

The two-byte address of the AES (Application Environment Services) parameter block. Use DPEEK and DPOKE for accessing this variable.

GCTRL**System variable**

The two-byte address of the AES (Application Environment Services) control block. Use DPEEK and DPOKE for accessing this variable.

GEMDOS**Function****Syntax:****GEMDOS(*function, parameter_list*)**

GEMDOS is the GEM interface to the disk operating system. This function allows access to the GEMDOS portion of the ST's operating system. It returns a 32-bit parameter. If no parameter is returned, or the returned value is not needed, the VOID statement may be used when calling GEMDOS functions. For more information and a complete list of GEMDOS routines and required parameters, refer to Appendix D.

Parameters:

function The function number of the called GEMDOS routine.

parameter_list A list of all required parameters to be passed to the called GEMDOS function. If a 16-bit word is required, the value may be passed with or without the 'W:' preface. 32-bit long words require a 'L:' preface.

Example:

```
a=GEMDOS(&h11) !Check printer status
IF a=1
    PRINT "Printer ready"
ELSE
    PRINT "Printer not ready"
ENDIF
```

Related Words:**BIOS, XBIOS**

GEMSYS
Function

Syntax: **GEMSYS** *number*
 GEMSYS(*number***)**

This function allows access to AES (Application Environment Services) routines.

Parameters:

number Function number for AES routine.

GET
Statement

Syntax: **GET** *x0,y0,x1,y1,a\$*

Abbreviation: **GET**

This function reads a rectangle from the screen and stores it as a bitpattern in a string variable.

Parameters:

x0 X coordinate of the upper left hand corner.

y0 Y coordinate of the upper left hand corner.

x1 X coordinate of the lower right hand corner.

y1 Y coordinate of the lower right hand corner.

a\$ String variable to hold bitpattern of defined area.

Example:

```
FOR ctr%=1 TO 5
    CIRCLE 320,100,ctr%*40
NEXT ctr%
GET 0,0,320,199,a$
PUT 120,0,a$,3
```

Related Words:

PUT

GET

Statement

Syntax:

GET[#[*num*[,*ctr*]]

This statement reads a record number for a Random access file.

Parameters:

num An integer expression between 0 and 99 which refers to the number of the data channel referenced with the OPEN statement.

ctr Record number to read. The maximum number of records in a Random access file is 65535. If the record number is not specified, the next record in the file is read.

NOTE: If you try to read information from a file which has just been placed there with a PUT statement, you must specify the record number to read, as the file pointer is positioned at the end of the file. Using GET will cause an "End of File" error under these circumstances.

Example:

```
OPEN "R",#1,"TEST.FIL",50
FIELD #1,20 AS A$,30 AS B$
FOR ctr%=1 TO 4
    INPUT "Name #1, Name #2:",v$,n$
    LSET a$=v$
    LSET b$=n$
    PUT#1,ctr%
NEXT ctr%
FOR ctr%=4 DOWNTO 1
    GET#1,ctr%
    PRINT a$,b$
NEXT ctr%
```

CLOSE#1

Related Words:

PUT

GINTIN

System Variable

This system variable contains the address of the AES integer input block.

GINTOUT

System Variable

This system variable contains the address of the AES integer output block.

GOSUB
Statement**Syntax:**

GOSUB proc_name[expression_list]
@ proc_name[expression_list]

Abbreviation:

GO

This statement causes the program to branch to a subroutine. The name of a procedure may begin with a number and can contain all alphanumeric symbols, dots, and the underline character.

Procedures can be nested. That is, a procedure may call another procedure. It is even possible for a procedure to call itself. This process is called recursion, or a recursive call.

The '@' sign may also be used in place of GOSUB. Unlike an abbreviation, this symbol will appear in the listed code.

Parameters:

proc_name The identifier of the procedure. Name may contain letters, numbers, dots, and the underline character.

expression_list A list of variables may be passed to the called procedure by the GOSUB call.

Example:

```
DO
    b%=0
    GOSUB proc_1
    GOSUB proc_2
    a=inp(2)
    EXIT IF a=13
LOOP
END
'
PROCEDURE proc_1
    PRINT "This is procedure 1"
RETURN
'
PROCEDURE proc_2
    INC b%
    PRINT "This is procedure 2"
    IF b%<3
        GOSUB proc_2
    ENDIF
RETURN
```

Related Words:**PROCEDURE****GOTO****Statement****Syntax:****GOTO *label*****Abbreviation:****GOT**

This statement permits an unconditional jump to another part of the program. GOTO may not be used to jump out of a Procedure.

This is a controversial point in programming. Many programmers feel that GOTOS are not necessary and should never be used.

GOTO is a part of the language, and as such is available to be used by the programmer. However, care should be used to avoid writing "spaghetti" code. (Code which jumps all over.) The best advice is to use GOTO sparingly.

Parameters:

label This is a character string, consisting of letters, numbers, dots, and dashes. A label can begin with a number. Since GFA BASIC does not use line numbers, all GOTO's must use a label for an address.

Example:

```
start:  
GOTO label_1  
label_2:  
PRINT "Line 2"  
GOTO start  
label_1:  
PRINT "Line 1"  
GOTO label_2
```

This is a good example of "spaghetti" code. Avoid this type of code at all costs. It's not only sloppy, but it's confusing. It also demonstrates that you don't understand program flow. Sloppy code is the mark of a novice programmer.

GRAPHMODE

Statement

Syntax:**GRAPHMODE** *number***Abbreviation:****G**

Sets up the proper mode for graphics functions.

Parameters:*number* A numeric expression representing the graphic mode to be used for Graphic statements.*number* = 1 **Replace Mode.**

The new picture or image is drawn on top of and replaces the previous picture or image.

number = 2 **Transparent Mode.**

The old picture or image can still be seen behind the new picture or image anywhere that the new image is drawn in the background color.

number = 3 **XOR Mode.**

Every element of the new picture is drawn except where a graphic dot is already present. Then the new graphic dot is XOR'd against the old one. XORing the new dot against the old dot twice leaves the old dot unchanged. This mode is useful for blinking images and simple animation.

number = 4 **Reverse Transparent.**

Essentially the same as Transparent Mode, except the new picture is shown as an inverse image.

Example:

```
DO
  IF k=1
    GRAPHMODE 1
  ELSE
    GRAPHMODE 3
  ENDIF
  MOUSE x,y,k
  CIRCLE x,y,50
  CIRCLE x,y,50
LOOP
```

HARDCOPY

Function

Syntax: HARDCOPY

Abbreviation: H

This function sends the contents of the screen to an attached printer. It is also possible to send the contents of the screen to the printer by pressing the <Alternate> and <Help> keys simultaneously.

Printing may be interrupted by pressing the <Alternate> and <Help> keys simultaneously.

Example:

```
FOR ctr%-1 TO 200 STEP 5
  CIRCLE ctr%,ctr%,20
NEXT ctr%
HARDCOPY
```

HEX\$
Function**Syntax:** **HEX\$(number)**

Converts a numeric value into a character string which contains the hexadecimal form of the value.

Parameters:

number Any integer between -2147483648 and +2147483647 in decimal, hexadecimal (&H), octal (&O), or binary (&X) format.

Example:

```
a=9  
b=&x1111  
PRINT HEX$(a)  
PRINT HEX$(b)  
PRINT HEX$(255)
```

Related Words:
BIN\$, OCT\$, STR\$

HIDEM
Function**Syntax:** **HIDEM****Abbreviation:** **HI**

Turns off (hides) the mouse pointer. Handy when it is desirable to not have the mouse interfere with a screen display. However, the mouse is still active.

Example:

```
HIDEM
CLS
PRINT "Press and Hold the LEFT Mouse Button to Draw"
PRINT "Press the RIGHT Mouse Button to exit"
DO
    EXIT IF MOUSEK=2
    IF MOUSEK=1
        PLOT MOUSEX,MOUSEY
    ENDIF
LOOP
```

Related Words:
SHOWM

HIMEM

System Variable

Syntax: **HIMEM**

System variable containing the address of the area in memory not required by GFA BASIC.

Related Words:
RESERVE, EXEC

IF
Statement**Syntax:** **IF** *conditional* [**THEN**]
 program block
 ELSE
 program block
 ENDIF

IF is a conditional branching statement which uses relational operators (=, <, >, <=, >=, <>) for evaluating a logical or boolean expression. (If condition is TRUE then execute the following statement.)

THEN is not a required portion of the statement and is only useful for clarification of the statement. (IF X=1 THEN do something.) Most experienced programmers skip this portion of the statement to save typing.

ELSE is an optional statement, which is used when the conditional statement is not satisfied. Useful when a branching action is desired.

ENDIF is required to close any IF statement. If ELSE is omitted from the block of code and the conditional is not satisfied, the entire block between IF and ENDIF is ignored.

Parameters:

conditional Any arithmetic, comparison or boolean operation.

program block Any program statement or group of statements to be executed if the conditional statement is satisfied.

Example:

```
FOR a%=1 to 100
  IF EVEN(a%)
    PRINT a%," is an even number"
  ELSE
    PRINT a%," is an odd number"
  ENDIF
  IF a%=99
    PRINT "Almost Done!"
  ENDIF
NEXT a%
```

IMP

Logical Function

Syntax:

IMP

Implication or conclusion, is only false when it is stated that something untrue can result from something that is true.

<u>Condition</u>	<u>EQV</u>	<u>Condition</u>	<u>Result</u>
-1	EQV	-1	-1
-1	EQV	0	0
0	EQV	-1	0
0	EQV	0	-1

Related Words:

AND, EQV, OR, NOT, XOR,

INC**Function****Syntax:** **INC variable****Abbreviation:** **IN**

Increase the value of a variable by one. Identical to **I=I+1**, but **INC** saves storage space and is much faster.

Parameters:**variable** Any numeric variable or numeric array variable.**Example:**

```
t=TIMER  
FOR ctr% = 0 TO 10000  
    INC a%  
NEXT ctr%  
PRINT (TIMER-t)/200  
a%=0  
t=TIMER  
FOR ctr% = 0 TO 10000  
    a%=a%+1  
NEXT ctr%  
PRINT (TIMER-t)/200
```

Related Words:**DEC**

INFOW**Function****Syntax:** **INFOW number,info_line****Abbreviation:** **INF**

This function creates an information line for a window to be opened with the **OPENW** statement.

An information line cannot be created for a window that has already been opened without an information line. If this function is called with the window open, the ST will lock up.

The information line may be changed after a window has been opened, providing the window was assigned an information line before the window was opened.

Parameters:

- number** A numeric expression, between 1 and 4, which determines the window number to which the information line is to be assigned.
- info_line** A character string which contains the information line.

Example:

```
INFOW 1,"This is window #1"
INFOW 2,"This is window #2"
OPENW 1
OPENW 2
PAUSE 300
INFOW 1 "This is STILL window #1"
CLOSEW 2
PAUSE 300
INFOW 1,"Now I claim to be #2!"
PAUSE 300
INFOW 3,"But, this is window 3!"
OPENW 3
PAUSE 300
CLOSEW 1
PAUSE 300
CLOSEW 3
```

Related Words:

OPENW, CLOSEW

INKEY\$

Function

Syntax: *variable=INKEY\$*

This function reads a character from the keyboard, as a character string consisting of none, one, or two, characters. If no key has been pressed since INKEY\$ was last called, a null string is returned.

If a key with an ASCII code assigned is pressed, the corresponding character is returned. The function keys and other keys, such as the <Help> key, return a 2 character string. The first character of this string is always a zero.

Parameters:

- variable** Any string variable.

Example:

```
DO
  a$=INKEY$
  PRINT LEN(a$)
  PRINT LEFT$(a$);";ASC(a$)
  PRINT RIGHT$(a$);";ASC(RIGHT$(a$))
LOOP
```

Press the <Control>, <Shift>, <Alternate> key combination to stop this demonstration program.

This function also presents an easy method of halting a program until a key is pressed.

```
PRINT "Press any key"
REPEAT
UNTIL INKEY$<> ""
PRINT "You pressed a key!"
```

INP

Function

Syntax: *variable* = INP(*x*)

variable = INP(#*x*)

This function reads one byte from a file or peripheral device. The program which called this function is halted until a byte is received.

INP(#*x*) will read one byte from a file on the *x* data channel.

Parameters:

variable Any numeric variable expression.

x An integer with a value between 0 and 4, representing an input device, according to the following table:

<u>Value</u>	<u>Input</u>	<u>Device</u>
0	LST:	(Printer Port)
1	AUX:	(Serial Port RS232)
2	CON:	(Keyboard)
3	MID:	(MIDI Port)
4	IKB:	(Input not allowed)

The input status of a device may be checked by using either of the following program lines:

status!=BIOS(1,x)

or

status!=INP?(x)

Where x represents the number of the device to be checked.

If status! = TRUE, at least one character is available.
If status! = FALSE, no characters are available.

Example:

This example reads input from the keyboard.

```
PRINT "Press <Return> to end"
DO
  EXIT IF a%=13
  a%=INP(2)
  PRINT CHR$(a%);
LOOP
```

This example will read a data channel.

```
OPEN "O",#1,"TEST.DAT"
PRINT #1,"TEST"
CLOSE #1
OPEN "I",#1,"TEST.DAT"
PRINT INP(#1)
CLOSE #1
```

Notice that the data file was created holding the string "TEST", but only the first byte, "T", was retrieved and printed on the screen.

Related Words:
INP?

INP?**Function****Syntax:** variable=INP?(number)

This function determines the status of an input device.

Parameters:

variable Any numeric variable expression. A zero is returned if no byte is present, if a byte is present, a value of -1 is returned.

x An integer with a value between 0 and 4, representing an input device, according to the following table:

<u>Value</u>	<u>Input</u>	<u>Device</u>
0	LST:	(Printer Port)
1	AUX:	(Serial Port RS232)
2	CON:	(Keyboard)
3	MID:	(MIDI Port)
4	IKB:	(Input not allowed)

Example:

```
DO
  EXITIF INP(2)=13
  IF INP?(2)
    a$=a$+STR$(INP(2))
  ENDIF
LOOP
PRINT a$
```

INPUT
Statement**Syntax:** INPUT ["text"],var_1,var_2**Abbreviation:** INP

This statement permits data to be assigned to variables from the keyboard during program execution. If the wrong information for the variable type assigned is entered, a bell sounds, and the data must be reentered.

Both the delete and backspace key function and may be used for editing the input. Also the right and left cursor keys may be used. The start and end of the text may be reached by using the up and down cursor keys.

Special characters can be entered in three different ways:

- By pressing the <Alternate> key at the same time as another key.
- By pressing the <Control> and <S> key simultaneously, followed by another key.
- By pressing the <Control> and <A> keys simultaneously, followed by the ASCII codes of the chosen character. (If the ASCII code is less than 26, you must then press any other non-numeric key.)

Parameters:

"text" A character string that will be shown on the screen as a prompt to the user. This text must always be enclosed in quotes. If a semicolon (;) is placed after the text, a question mark will be placed immediately to the right of the text.

If a comma separates the text from the variable, information will be entered immediately following the text.

var_1... Any variable name. If characters are to be entered, a character string must be assigned. More than one variable may be assigned for multiple inputs, but each variable must be separated by a comma, and the data input to these variables must also be separated. If a character string is input, it may be up to 255 characters long.

If <Return> is pressed before all the required data is entered, a question mark will be displayed, informing you that more entries are required.

Example:

```
INPUT "Enter your name: ",name$  
INPUT "How old are you";age  
INPUT "What city and state do you live in";city$,state$  
PRINT name$  
PRINT age  
PRINT city$,state$
```

Related Words:

INPUT#

INPUT#
Statement**Syntax:** **INPUT# ch, var_list**

This statement permits data to be assigned to variables being read from the disk drive. If the wrong information for the variable type assigned is read, an error message will appear.

Parameters:

- ch** The channel number from which the data is to be read. Assigned by the OPEN statement.
- var_list...** Any variable name. If characters are to be entered, a character string must be assigned. More than one variable may be assigned for multiple inputs, but each variable must be separated by a comma, and the data input to these variables must also be separated. If a character string is input, it may be up to 255 characters long.

Example:

```
INPUT "Enter your name",name$  
INPUT "Enter your age",age  
OPEN "O",#1,"SAMPLE.DAT"  
PRINT #1,name$  
PRINT #1,age  
CLOSE #1  
name$=""  
age=0  
CLS  
OPEN "I",#1,"TEST.DAT"  
INPUT#1,name$  
INPUT#1,age  
CLOSE #1  
PRINT name$  
PRINT age
```

Related Words:**INPUT, WRITE#**

INPUT\$
Function**Syntax:** **INPUT\$(number[,#n])**

Reads a specified number of characters from the keyboard or from a file as a string.

Parameters:**number** An integer with a value between 0 and 32767.**#n** An optional portion of the function, which if not used, causes the string of number characters to be read from the keyboard. If this parameter is specified, the string is read from the data channel specified by n.

This function will read all characters from a file, including those which would normally terminate input. INPUT\$ may be used for reading Dexos™ and NEOchrome™ pictures.

Example:

```
PRINT "Enter 6 characters"
PRINT INPUT$(6)
OPEN "O",#1,"TEST.DAT"
PRINT #1,"TEST"
CLOSE #1
OPEN "I",#1,"TEST.DAT"
PRINT INPUT$(2,#1)
CLOSE #1
```

INSTR
Function**Syntax:**

INSTR([number],svar_1,svar_2)
INSTR([svar_1,svar_2,[number]])

Searches for a designated character string within another string. If found, the position of the found string is returned, if not, a zero is returned.

Parameters:

- number** A numeric expression indicating the position within the character string that the position to begin the search. If no number is indicated, the search begins with the first character in the string.
- svar_1** This is the character string to be searched.
- svar_2** This is the search string. Any character string expressions are permitted.

Example:

```
a$="GFA BASIC is great!"  
b$="great"  
PRINT INSTR(a$,b$)  
PRINT INSTR(4,a$,b$)  
PRINT INSTR(a$,"abc")
```

INT
Function**Syntax:**

INT(numeric_variable)

Returns the integer that is less than or equal to a numeric integer.

Parameters:

- numeric_variable** Any numeric expression. If the numeric expression is a negative number, the value returned is the next lowest value.

Example:

```
a=PI  
PRINT a  
PRINT INT(a)  
PRINT INT(2.00001)  
PRINT INT(12345)
```

INTIN**System Variable**

Holds the two-byte address of the VDI (Virtual Device Interface) input parameter block.

INTOUT**System Variable**

Holds the two-byte address of the VDI (Virtual Device Interface) output parameter block.

KILL**Command**

Syntax: **KILL "filespec"**

Deletes a disk file. Deletes only one file at a time. This command accepts "?" and "*" for wild cards. Use caution if you use wild cards as part of your file name for removing a file.

Parameters:

filespec Name of the file to remove from a disk. *filespec* need not be enclosed in quotes. Hierarchical file system may be used.

Example:

```
OPEN "O",#1,"JUNK.FIL"
CLOSE #1
FILES
KILL "JUNK.FIL"
PRINT
FILES
```

LEFT\$
Function**Syntax:** **LEFT\$(string,[int])**

Used to extract one, or a specified number of characters from a string beginning with the leftmost character.

Parameters:

string A character string or a string variable.

int An integer expression. If no value is defined, only the leftmost character is returned. If the value is larger than the string, the entire string is returned.

Example:

```
a$="Forescore and seven years ago"
PRINT LEFT$(a$)
PRINT LEFT$(a$,3)
PRINT LEFT$(a$,255)
```

Related Words:

RIGHT\$, MID\$

LEN
Function**Syntax:** **LEN(string_var)**

This function returns the length of a string variable.

Parameters:

string_var Any character string expression.

Example:

```
a$="Test String"
PRINT LEN(a$)
PRINT LEN(a$)+" GFA BASIC"
a%=LEN(a$)
PRINT a%
```

LET
Statement**Syntax:** **LET variable = constant**

This statement may be used to assign values to variables. It permits some reserved words to be used as variables.

Parameters:**variable** Any legal variable name or type.**constant** Any constant expression or value to be assigned to a variable.**Example:**

```
LET a=12345
LET a$="String Variable"
PRINT a
PRINT a$
```

LINE
Function**Syntax:** **LINE x0,y0,x1,y1****Abbreviation:** **LI**

Connects two points with a straight line. The origin for the coordinates is the top left corner of the screen. The color used for drawing the line is defined by the COLOR statement. The attributes of the line may be defined by using the DEFLINE function.

This statement is identical to the DRAW TO statement.

Parameters:**x0** Integer value of first x coordinate.**y1** Integer value of first y coordinate.**x1** Integer value of second x coordinate.**y1** Integer value of second y coordinate.

Example:

```
LINE 0,0,639,199  
LINE 639,0,0,199
```

Related Words:

DEFLINE

LINE INPUT

Statement

Syntax:

LINE INPUT [text]var_1,var_2

Abbreviation:

LI INPUT

Allows a string to be entered during program execution. A comma is not regarded as a separator. Only RETURN, ASC(13), is used as a separator.

Parameters:

text Any string to be shown on the screen as a prompt. If the text prompt is followed by a semicolon, a question mark will follow the prompt. Otherwise, a space will be printed following the prompt string.

var_1, var_2 Any string variable or list of variables.

Example:

```
LINE INPUT a$  
LINE INPUT a$,b$,c$  
LINE INPUT "Enter your name",name$
```

Related Words:

INPUT, INPUT#, LINE INPUT#

LINE INPUT#**Statement****Syntax:****LINE INPUT# *number*, *var_1*,*var_2*****Abbreviation:****LI INPUT****Parameters:**

number The channel number assigned by the OPEN statement.

var_1, *var_2* Any string variable or list of variables.

Example:

```
LINE INPUT #1, a$  
LINE INPUT #1,a$,b$,c$
```

Related Words:**INPUT, INPUT#, LINE INPUT**

LIST**Command****Syntax:****LIST "filename"****Abbreviation:****LIS**

Stores the program currently in memory in ASCII format. This command may be entered by a program line, in direct mode, or by selecting the *SAVE,A* function from the Editor menu.

Parameters:

filename Any legal filename. The hierarchical file system may be used. If this is an empty string, the listing will be shown on the screen.

If a program segment is to be merged with other program segments, it must first be saved in ASCII format.

Example:

```
PRINT "LINE 1"  
PRINT "LINE 2"  
LIST ""  
LIST "TEST.ASC"
```

LLIST
Command**Syntax:** **LLIST****Abbreviation:** **LL**

Sends a listing of the program currently in memory to an attached line printer. The listing may be interrupted by turning off the printer. After about 30 seconds, control will return to the editor.

LOAD
Command**Syntax:** **LOAD "filename"****Abbreviation:** **LOA**

Loads a program from disk into memory. This command may be entered under program control, in direct mode, or by selecting LOAD from the Editor Menu.

Parameters:

filespec Any legal filename. The hierarchical file system may be used. If no extension to the filename is specified, the .BAS extension is assumed. The file name need not be contained in quotes.

Related Words:**SAVE**

LOC
Function**Syntax:** **LOC([#]channel_number)**

This function returns the location of the file pointer for a previously opened file. Each data channel has a file pointer which points to a byte in each file. LOC returns the displacement in bytes from the beginning of the file. This function may only be used with disk files.

Parameters:

[#]channel_number An integer expression between 0 and 99 which represents the number assigned to a file with the OPEN statement.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT #1,"ABCDEFGHIJKL"
SEEK #1,3
PRINT LOC(#1)
CLOSE #1
```

LOCAL
Statement**Syntax:** **LOCAL variable****Abbreviation:** **LOC**

Declares a variable name to be "local", or to apply only within a subroutine. A variable name declared as local to a subroutine may be used outside the subroutine without conflict. No array variables may be used as local variables.

Parameters:

variable Any valid variable name.

Example:

```
A%=12345
PRINT a%
GOSUB subroutine
PRINT A%
'
PROCEDURE subroutine
LOCAL a%
a%=1
PRINT a%
RETURN
```

Related Words:

GOSUB, PROCEDURE, RETURN

LOF

Function

Syntax: **LOF([#]channel_number)**

Determines the length of the file with the channel number specified in the OPEN statement. This function may only be used with disk files.

Parameters:

channel_number An integer expression between 0 and 99 which represents the channel number designated by the OPEN statement.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT LOF(#1)
PRINT #1,"ABCDEFGHIJKL"
PRINT LOF(#1)
CLOSE #1
```

LOG
Function**Syntax:** **LOG(*number*)**

Determines the natural (base e) logarithm of a number.

Parameters:*number* Any numeric expression with a value greater than zero.**Example:**

```
a=2.34567  
PRINT LOG(a)  
PRINT LOG(3.456)
```

Related Words:
LOG10

LOG10

Function

Syntax: LOG10(*number*)

Determines the base 10 logarithm of a number.

Parameters:*number* Any numeric expression with a value greater than zero.**Example:**

```
a=2.34567  
PRINT LOG10(a)  
PRINT LOG10(1000)
```

Related Words:

LOG

LPEEK

Function

Syntax: LPEEK (*address*)

Returns the contents of 4 consecutive bytes of memory as a long word.

Parameters:*address* An even numeric expression which specifies an address in memory.**Example:**

```
a$="A"  
desc=ARRPTR(a$)  
addr=LPEEK(d)  
length=DPEEK(d+4)  
ascii=PEEK(a)  
PRINT "Descriptor Address: ";desc  
PRINT "Address : ";addr  
PRINT "Length : ";length  
PRINT "ASCII Value : ";ascii
```

Related Words:

PEEK, DPEEK

LPOKE
Function**Syntax:** **LPOKE address, number****Abbreviation:** **LP****Stores 4 bytes into an area of main memory at a specified address.****Parameters:****address** A even numeric expression which specifies an address in memory.**number** A numeric expression between -2147483648 and +2147483648.**Example:**

```
a$="A"  
ptr=ARRPTR(a$)  
DPOKE ptr+4,4  
z=VARPTR(a$)  
LPOKE z,111638594  
PRINT a$  
POKE z,67  
PRINT a$
```

Related Words:

POKE, DPOKE, SPOKE, SLPOKE

LPOS
Function**Syntax:** **LPOS (number)**

Returns the number of the column in which the print head of an attached printer is located. This value may not correspond to the actual physical position of the print head in every case, as the function is sensitive to the number of characters printed. Characters such as a carriage return (ASCII 13), line feed (ASCII 10) and backspace (ASCII 8) are counted as characters.

Parameters:

number Any numeric expression.

Example:

```
FOR ctr%=1 TO 600
LPRINT CHR$(65)
IF LPOS(1)=60
LPRINT
ENDIF
NEXT ctr%
```

This example prints 10 lines of the letter "A", each line 60 characters wide.

LPRINT Statement

Syntax: **LPRINT** *expression*

Abbreviation: **LPR**

Sends data to an attached printer. This statement is similar to the PRINT statement, and may be executed from within a program or in direct mode.

Parameters:

expression A valid numeric or string expression, or group of expressions.

Example:

```
LPRINT "Sample line to printer"
a=12345
b=10.78
a$="Another sample line for the printer."
LPRINT a
LPRINT USING "##.##",b
LPRINT USING "##.##",PI
LPRINT a$
```

Related Words:

PRINT

LSET
Function**Syntax:** **LSET variable = expression****Abbreviation:** **LS**

Left justifies a string variable. Normally used in conjunction with FIELD when creating a Random access file.

Parameters:

variable Any legal string variable name. This variable should be previously defined with a template containing the maximum number of characters permitted in the string to be justified.

expression Any string expression. In the case of numeric values, the functions MKI\$, MKS\$, MKL\$, or MKD\$ must be used to transform the data into a string before calling this function. If the total characters in this expression is greater than the maximum allowable number of characters, the string is truncated. If the string is shorter, then the unused characters will contain spaces.

Example:

```
a$="AAAAAAAAA"  
b$=SPACE$(9)  
c$="Sample"  
LSET a$=c$  
LSET b$=c$  
PRINT c$  
PRINT a$  
PRINT b$  
LSET b$="This string is truncated"  
PRINT b$
```

Related Words:**RSET**

MAX
Function**Syntax:** **MAX (expression_list)**

This function returns the greatest value (or numerically largest string) from a list of expressions. This function is similar to using the comparator ">" (greater than) to logically compare expressions.

Parameters:

expression_list Any numeric or string expression. All expressions in the list must be the same type. In other words, mixing string variables and numeric variables is not permitted.

Example:

```
a=100  
b=50  
PRINT MAX(a,b)  
a$="AAAAA"  
b$="ZZZZZ"  
PRINT MAX(a$,b$)
```

Related Words:

MIN, > (greater than), < (less than)

MENU
Statement**Syntax:** **MENU field**
 MENU item_number,integer**Abbreviation:** **ME**

Creates and modifies the menu bar.

Parameters:

field A one dimensional string array which contains the text of the menu bar. The menu title and items of each pull down menu must be entered consecutively. An empty string must follow each menu title and it's respective pulldown menu.

item_number Item number in the menu.

integer An integer value between 0 and 3.

Menu items may be altered by using MENU item_number, integer. This is similar to the index of a string array.

- | | |
|----------------------------|--|
| MENU <i>item_number</i> ,0 | Removes a check mark from in front of the menu item. |
| MENU <i>item_number</i> ,1 | Places a check mark in front of a menu item. In order to do this, it is necessary that you leave space in front of the menu item when you define it. |
| MENU <i>item_number</i> ,2 | Writes the menu item in ghost lettering and disables selection. |
| MENU <i>item_number</i> ,3 | Writes the menu item in normal lettering and enables selection. |

Example:

```
DIM menu_strip$(50)
FOR ctr%=0 TO 50
    READ menu_strip$(ctr%)
    EXIT IF menu_strip$(ctr%)="***"
NEXT ctr%
menu_strip$(ctr%)=""
menu_strip$(ctr%+1)=""
DATA Desk, Menu Test
DATA 1,2,3,4,5,6,""
DATA File, Line 1, Line 2, Line 3,""
DATA Help, Help 1, Help 2,""
DATA Edit, Edit 1, Edit 2, Edit 3, Quit,""
DATA ***
MENU menu_strip$()
ON MENU GOSUB menu_routine
DO
    ON MENU
LOOP

PROCEDURE menu_routine
    PRINT "Menu selection: ";
    PRINT menu_strip$(MENU(0))
    IF menu_strip$(MENU(0))=" Quit"
        END
    ENDIF
    MENU OFF
RETURN
```

Related Words:

MENU KILL, MENU OFF, ON MENU GOSUB

MENU KILL

Statement

Syntax: MENU KILL

Abbreviation: ME KILL

Deactivates a previously defined menu

Related Words:

MENU, MENU, MENU OFF, ON MENU GOSUB

MENU OFF

Statement

Syntax: MENU OFF

Abbreviation: ME OFF

Any time a menu item is selected, the menu title will be displayed in inverse print (generally, white characters on a black background). This statement returns the display of the menu item to normal mode.

Example:

(See MENU)

Related Words:

MENU, MENU KILL, ON MENU GOSUB

MENU
Function**Syntax:** **MENU(*number*)**

After ON MENU has been triggered by one of the specified events, this function makes it possible to determine the values that were returned.

Parameters:*number* An integer value.

<u>Integer</u>	<u>Meaning</u>
0	The number of the selected pull down menu item, as specified in the menu definition, is returned. The number corresponds to the field of the string array occupied by the selected menu item.
18	GEM messages which indicate which menu event has occurred. GEM messages are found in MENU(1) through MENU(8). The interpretation of the messages depends on the value returned by MENU(1).

<u>Value of MENU(1)</u>	<u>Meaning</u>
10	User has selected a menu item. MENU(4) indicates which title, MENU(5) indicates the item under the title which has been selected. Not normally used by GFA BASIC. ON MENU works better.
20	User has done something that requires redrawing the window, or a portion of the window. MENU(4) tells which window needs to be redrawn. MENU(5), MENU(6), MENU(7), and MENU(8) indicate the x-coordinate, y-coordinate, width and height or the rectangle to be redrawn. Normally, GFA BASIC users only see this message when a window is first opened.

- 21 Window topped. A partially hidden window has been brought to the top. MENU(4) indicates which window. This normally triggers a redraw message. This message is not normally seen by GFA BASIC users.
- 22 Window closed. MENU(4) indicates which window the user has selected to close.
- 23 Window fulled. MENU(4) indicates which window the user has selected to make full.
- 24 User has selected an arrow or the shaded portion of the sliders at the bottom or right side of the window. MENU(4) indicates which window was selected. MENU(5) indicates the user's selection:
 - 0 Page up
 - 1 Page down
 - 2 Row up
 - 3 Row down
 - 4 Page left
 - 5 Page right
 - 6 Column left
 - 7 Column right
- 25 Bottom slider selected and moved. MENU(4) indicates window selected. MENU(5) contains the new slider location, with a number between 0 (extreme right) and 1000 (extreme left).
- 26 Right slider selected and moved. MENU(4) indicates window selected. MENU(5) contains the new slider location, with a number between 0 (top) and 1000 (bottom).
- 27 Window resized. Window moved. MENU(4) indicates window selected. MENU(5), MENU(6), MENU(7) and MENU(8) indicate the new x-coordinate, y-coordinate, width, and height of the window.

- 28 Window moved. MENU(4) indicates window selected. MENU(5), MENU(6), MENU(7) and MENU(8) indicate the new x-coordinate, y-coordinate, width, and height of the window.
- 9 Contains the content of INTOUT(9), indicating which event caused the ON MENU command.
- 1 Key pressed.
 - 2 Mouse button pressed.
 - 4 Mouse entered or exited one of two predefined rectangles.
 - 8 Mouse entered or exited second of two predefined rectangles.
 - 16 GEM message received.
 - 32 Timer setting expired.
- 10 & 11 Returns the x and y coordinates of the mouse (INTOUT(1) and INTOUT(2)).
- 12 Status of mouse buttons:
- 0 No mouse button pressed.
 - 1 Left mouse button pressed.
 - 2 Right button pressed.
 - 3 Both buttons pressed.
- 13 Status of special keyboard keys:
- 0 No special key pressed.
 - 1 Right shift key pressed.
 - 2 Left shift key pressed.
 - 4 Control key pressed.
 - 8 Alternate key pressed.
- 14 Scan code of key returned in the high byte, ASCII code in low byte. INTOUT(5)
- 15 Number of mouse clicks. INTOUT(6)
- 16 Address of the object in the menu object tree.

Example:

```
IF MENU(14) and 255  
  key$=CHR$(MENU(14))  
ELSE  
  key$=MKI$(MENU(14))  
ENDIF
```

This short example makes sure that a letter key has been pressed and not a function or special character key.

Related Words:

MENU, MENU OFF, MENU KILL, ON MENU GOSUB

MID\$

Function

Syntax:

MID\$(string_expression,position,[length])

Returns a specified number of characters from within a string.

Parameters:

string_expression A character expression or string variable.

position An offset, measured from the leftmost character within the string expression, from which MID\$ begins reading characters.

[length] Number of characters from the indicated position to read. If the length specified is larger than the number of characters from the position selected to the end of the string, the characters from the selected position to the end of the string are returned.

This is an optional parameter, and if not specified, all characters from the specified position to the end of the string are returned.

Example:

```
a$="This is a string."
PRINT MID$(a$,0,4)
PRINT MID$(a$,5,3)
PRINT MID$(a$,8)
```

Related Words:

LEFT\$, RIGHT\$, MID\$=

MID\$=

Function

Syntax: **MID\$(*string_expression,position,[length]*)=*expression_2***

Permits parts of strings to be modified.

Parameters:

string_expression Any character expression or string variable. This string will be used to modify another string.

position An offset, measured from the leftmost character within the string expression. At this point MID\$= begins replacing characters.

[length] Number of characters from the indicated position to replace. If the length specified is larger than the number of characters from the position selected to the end of the string, the characters from the selected position to the end of the string are returned.

This is an optional parameter, and if not specified, all characters from the specified position to the end of the string are replaced. The length of the string variable is not changed.

expression_2 Text string to be changed.

Example:

```
a$="ABCDEF"
MID$(a$,3,4)="XZ"
PRINT a$
```

Related Words:

LEFT\$, RIGHT\$, MID\$

MIN**Function****Syntax:****MIN (expression_list)**

This function returns the lowest value (or numerically smallest string) from a list of expressions. This function is similar to using the comparator “<” (less than) to logically compare expressions.

Parameters:

expression_list Any numeric or string expression. All expressions in the list must be the same type. In other words, mixing string variables and numeric variables is not permitted.

Example:

```
a=100  
b=50  
PRINT MIN(a,b)  
a$="AAAAA"  
b$="ZZZZZ"  
PRINT MIN(a$,b$)
```

Related Words:

MAX, > (greater than), < (less than)

MKD\$
Function**Syntax:** **MKD\$(*numeric_value*)**

Transforms a numeric value into a MBASIC compatible 8-byte character string.

Parameters:*numeric_value* Any numeric expression.**Example:**

```
a=.1234
PRINT MKD$(a)
```

Related Words:

MKI\$, MKL\$, MKS\$, MKF\$

MKDIR
Statement**Syntax:** **MKDIR "directory_name"****Abbreviation:** **MK**

Creates a new directory. It is possible to use the hierarchical file system when choosing a name. The new directory will be created within the current directory, if the hierarchical file system is not used.

Parameters:*directory_name* Name and path to the directory to be created.**Example:**

```
MKDIR "A:\TEST1"
MKDIR "A:\TEST1\TEST2"
MKDIR "A:\TEST1\TEST2\TEST3"
MKDIR "TEST4"
FILES "A:\*.*"
FILES "A:\TEST1\
FILES "A:\TEST1\TEST2\"
```

Related Words:

CHDIR

MKF\$
Function**Syntax:** **MKF\$(*numeric_value*)**

Transforms a numeric value into a 6-byte number in GFA BASIC format.

Parameters:*numeric_value* Any numeric expression.**Example:**

```
a=.1234
PRINT MKF$(a)
```

Related Words:
MKI\$, MKL\$, MKS\$, MKD\$

MKI\$
Function**Syntax:** **MKI\$(*numeric_value*)**

Transforms a 16-bit integer numeric value into a 2-byte string.

Parameters:*numeric_value* Any numeric expression.**Example:**

```
a=3456
PRINT MKI$(a)
```

Related Words:
MKD\$, MKL\$, MKS\$, MKF\$

MKL\$
Function**Syntax:** **MKL\$(*numeric_value*)**

Transforms a 32-byte integer numeric value into a 4-byte string.

Parameters:*numeric_value* Any numeric expression.**Example:**

```
a=1234567890
PRINT MKD$(a)
```

Related Words:
MKI\$, MKD\$, MKS\$, MKF\$

MKS\$
Function**Syntax:** **MKS\$(*numeric_value*)**

Transforms a numeric value into a 4-byte format number compatible with Atari BASIC.

Parameters:*numeric_value* Any numeric expression.**Example:**

```
a=.1234
PRINT MKS$(a)
```

Related Words:
MKI\$, MKL\$, MKD\$, MKF\$

MOD
Function**Syntax:** *var_1 MOD var_2*

Computes the remainder (modulo) after the first value is divided by the second. The remainder is then rounded up or down to the nearest integer value.

Parameters:*var_1* Any numeric expression.*var_2* Any numeric expression.**Example:**

PRINT 8 MOD 5

Related Words:
INT, FIX, FRAC, DIV, /**MONITOR**
Command**Syntax:** MONITOR(*number*)**Abbreviation:** MON

Calls a monitor resident in memory or an extension to GFA BASIC.

Parameters:*number* This integer value is passed to the monitor or commands extension in register D0.

MOUSE
Statement**Syntax:** **MOUSE** *x,y,k***Abbreviation:** **M**

Determines the mouse position and the status of the mouse buttons.

Parameters:*x* X coordinate of mouse.*y* Y coordinate of the mouse.*k* Mouse button state:

- 0 No button pressed.
- 1 Left button pressed.
- 2 Right Button pressed.
- 3 Both buttons pressed.

Example:

```
DO
  MOUSE x%,y%,k%
  PRINT at(1,1);
  PRINT "X=";x%
  PRINT "Y=";y%
  PRINT "Mouse Button=";k%
LOOP
```

Related Words:

MOUSEX, MOUSEY, MOUSEK

MOUSEK**Statement****Syntax:** *variable* = **MOUSEK**

Returns the status of the mouse buttons.

Parameters:*variable* Any integer variable expression.

<u>Number Returned</u>	<u>Meaning</u>
0	No button pressed.
1	Left button pressed.
2	Right Button pressed.
3	Both buttons pressed.

Example:

```
DO
  x% = MOUSEX
  y% = MOUSEY
  k% = MOUSEK
  PRINT at(1,1);x%,y%,k%
LOOP
```

Related Words:

MOUSE, MOUSEX, MOUSEY

MOUSEX**Statement****Syntax:** *variable* = **MOUSEX**

Returns the X-coordinate of the mouse position.

Parameters:*variable* Any numeric variable

Example:

```
DO  
  x% = MOUSEX  
  y% = MOUSEY  
  k% = MOUSEK  
  PRINT at(1,1)x%,y%,k%  
LOOP
```

Related Words:

MOUSE, MOUSEY, MOUSEK

MOUSEY

Statement

Syntax:

variable = MOUSEY

Returns the Y-coordinate of the mouse position.

Parameters:

variable Any numeric variable

Example:

```
DO  
  x% = MOUSEX  
  y% = MOUSEY  
  k% = MOUSEK  
  PRINT at(1,1)x%,y%,k%  
LOOP
```

Related Words:

MOUSE, MOUSEX, MOUSEK

MUL

Function

Syntax: **MUL var_1,var_2****Abbreviation:** **MU**

Multiplies two numeric expressions. This is identical to the standard multiplication function (*var * var*), but is much faster.

Parameters:**var_1** Any numeric expression.**var_2** Any numeric expression.**Example:**

```
DIM a%(10000)
ARRAYFILL A%(),5
t=TIMER
FOR ctr%=1 TO 10000
    MUL a%(ctr%),5
NEXT ctr%
PRINT (TIMER-t)/200
t=TIMER
FOR ctr%= 1 TO 10000
    A%(ctr%)=a%(ctr%)*5
NEXT ctr%
PRINT (TIMER-t)/200
```

Related Words:

ADD, DIV, SUB

NAME**Function****Syntax:** **NAME "oldfile" AS "newfile"****Abbreviation:** **NA**

Changes the name of a file on a disk. Does not move files between disks.

Parameters:**oldfile** Any legal file name existing on a disk.**newfile** Any legal file name to rename oldfile as.**Example:**

```
OPEN "O",#1,"OLDFILE.TXT"  
CLOSE #1  
FILES  
NAME "OLDFILE.TXT" AS "NEWFILE.TXT"  
FILES
```

NEW**Command****Syntax:** **NEW**

Erases the program presently in memory.

Example:

```
PRINT "Test line"  
NEW
```

This is not a useful program example. It removes itself from memory as soon as it is run.

NOT
Logical Operator**Syntax:** **NOT** *condition*

Reverses an outcome or logical operation, making a true condition false, and vice versa.

Parameters:*condition* Any logical expression.

	<u>NOT</u>	<u>Condition</u>	<u>Result</u>
NOT	-1	(TRUE)	0 (FALSE)
NOT	0	(FALSE)	-1 (TRUE)

Example:

```
a=9
IF NOT(a=4)
    PRINT "NOT' failed the IF THEN test"
ELSE
    PRINT "NOT' passed the IF THEN test"
ENDIF
```

Related Words:

AND, OR, XOR, IMP, EQV, IF, WHILE, REPEAT

OCT\$
Function**Syntax:** OCT\$(*variable*)

Converts a numerical expression into a character string in octal form. The numerical expression can be in any form (hexidecimal, decimal or binary) between -2147483648 and +2147483647.

Parameters:

variable Any numeric expression between -2147483648 and +2147483647.

Example:

```
a=12
PRINT OCT$(a)
PRINT OCT$(1234)
PRINT OCT$(&H20)
```

Related Words:
BIN\$, HEX\$

ODD
Function**Syntax:** ODD(*variable*)

Determines whether a numerical expression is an odd number. The value -1 (TRUE) is returned for an odd number, zero (FALSE) for an even number.

Parameters:

variable Any numeric expression.

Example:

```
FOR ctr%=1 TO 100
  IF ODD(ctr%)
    PRINT ctr%
  ENDIF
NEXT ctr%
```

Related Words:
EVEN

ON BREAK

Statement

Syntax:

```
ON BREAK
ON BREAK CONT
ON BREAK GOSUB proc_name
```

These three statements determine how a program handles the break command. The break command can be sent by simultaneously pressing the <Control>, <Shift>, and <Alternate> keys.

The ON BREAK CONT statement deactivates this key sequence, while ON BREAK re-enables the sequence.

ON BREAK GOSUB *proc_name* makes it possible to jump to a procedure by pressing the "Break" key combination. This procedure may then carry out any necessary action. ON BREAK again deactivates the ON BREAK GOSUB statement, and allows the "break" keys to function normally to interrupt a running program.

Parameters:

proc_name Any valid name for a procedure.

Example:

```
ON BREAK GOSUB proc_1
PRINT "Press right mouse button to exit."
DO
  EXIT IF k% = 1
  k% = MOUSEK
LOOP
'
PROCEDURE proc_1
  PRINT "Break keys disabled!"
RETURN
```

Run this sample program, and try pressing the <Control>, <Shift>, and <Alternate> key combinations.

ON ERROR
Statement**Syntax:****ON ERROR**
ON ERROR GOSUB *proc_name*

Enables an error trapping routine to detect any errors and take the appropriate action by jumping to the procedure named by the ON ERROR GOSUB statement. ON ERROR GOSUB must be re-enabled after each error message.

ON ERROR deactivates this error trap.

Parameters:

proc_name Any valid procedure name.

Example:

```
ON ERROR GOSUB error_handler
PRINT SQR(1)
PRINT 3/0
ON ERROR
PRINT SQR(1)

PROCEDURE error_handler
  PRINT "Error number: ";ERR
  ON ERROR GOSUB error_handler
  RESUME NEXT
RETURN
```

Related Words:

RESUME

ON MENU**Function****Syntax:** **ON MENU GOSUB procedure**

This function defines the procedure which handles menu selection.

Parameters:

procedure Any valid procedure name.

Example:

```
DIM menu_strip$(50)
FOR ctr% = 0 TO 50
    READ menu_strip$(ctr%)
    EXIT IF menu_strip$(ctr%) = "***"
NEXT ctr%
menu_strip$(ctr%) = ""
menu_strip$(ctr% + 1) = ""
DATA Desk, Menu Test
DATA 1,2,3,4,5,6,""
DATA File, Line 1, Line 2, Line 3, ""
DATA Help, Help 1, Help 2, ""
DATA Edit, Edit 1, Edit 2, Edit 3, Quit, ""
DATA ***
MENU menu_strip$()
ON MENU GOSUB menu_routine
DO
    ON MENU
LOOP
'
PROCEDURE menu_routine
    PRINT "Menu selection: ";
    PRINT menu_strip$(MENU(0))
    IF menu_strip$(MENU(0)) = " Quit"
        END
    ENDIF
    MENU OFF
RETURN
```

In a practical program, PROCEDURE menu_routine would evaluate the menu selections and take whatever action appropriate, just as it does in this example when "Quit" is selected. Notice the double space before each entry under a heading. This allows space for a checkmark in front of each name.

Related Words:

ON MENU KEY, ON MENU MESSAGE, ON MENU IBOX, ON MENU OBOX, ON MENU BUTTON

ON MENU KEY**Function****Syntax:****ON MENU KEY GOSUB *procedure***

This function defines the procedure which handles keyboard input.

Parameters:

procedure Any valid procedure name.

Example:

```
ON MENU KEY GOSUB menu_key
DO
  'This loops does nothing but wait.
LOOP

PROCEDURE menu_key
  PRINT "Key number ";MENU(10);"Has been pressed"
  PRINT "Status of Alternate keys is ";MENU(13)
  RETURN
```

Related Words:

ON MENU, ON MENU MESSAGE, ON MENU IBOX, ON MENU OBOX, ON MENU BUTTON

ON MENU MESSAGE

Function

Syntax: **ON MENU MESSAGE GOSUB procedure**

This function defines the handling routine for GEM messages. Refer to MENU for more information on GEM messages.

Parameters:

procedure Any valid procedure name.

Example:

```
attr=WINTAB+2
DPOKE attr,&HFFF
TITLEW 1,"Menu Message Demo"
OPENW 1
FULLW 1
ON MENU MESSAGE GOSUB menu_message
DO
'This loop does nothing but wait
LOOP
```

```
PROCEDURE menu_message
PRINT "GEM message: "
m=MENU(1)
IF m=20
  msg$="Window Redraw"
ENDIF
IF m=21
  msg$="Window Topped"
ENDIF
IF m=22
  msg$="Window Closed"
ENDIF
IF m=23
  msg$="Window Fulled"
ENDIF
IF m=24
  msg$="Window Arrow"
ENDIF
IF m=25
  msg$="Horizontal Slider"
ENDIF
IF m=26
  msg$="Vertical Slider"
ENDIF
```

```
IF m=27
  msg$="Window Sized"
ENDIF
IF m=28
  msg$="Window Moved"
ENDIF
PRINT m;" ";msg$
RETURN
```

Related Words:

ON MENU KEY, ON MENU, ON MENU IBOX, ON MENU OBOX, ON MENU BUTTON

ON MENU IBOX**ON MENU OBOX**

Function

Syntax:

ON MENU IBOX a,x,y,b,h GOSUB procedure
ON MENU OBOX a,x,y,b,h GOSUB procedure

This function defines the procedure to determine when the mouse moves into (IBOX) or out of (OBOX) a predefined rectangle.

Parameters:

- a Defines which one of two possible, predefined independent rectangles to use.
- x X-coordinate of the upper left corner of the rectangle.
- y Y-coordinate of the upper left corner of the rectangle.
- w Width of the rectangle.
- h Height of the rectangle.

Example:

```
ON MENU IBOX 1,100,100,200,100 GOSUB ibox_procedure
DO
  'This loop does nothing but wait for a menu selection
LOOP
```

```
PROCEDURE ibox_procedure
  DEFMOUSE 2
  ON MENU OBOX 1,100,100,200,100 GOSUB obox_procedure
  PRINT "Mouse x = ";MENU(10); and Mouse y =";MENU(11)
  PRINT MOUSE Button = ";MENU(11)
RETURN
```

```
PROCEDURE obox_procedure
  DEFMOUSE 0
  ON MENU IBOX 1,100,100,200,100 GOSUB ibox_procedure
RETURN
```

Related Words:

ON MENU KEY, ON MENU, ON MENU MESSAGE, ON MENU BUTTON

ON MENU BUTTON

Function

Syntax:

ON MENU BUTTON *max_clicks, butn_mask, butn_status* GOSUB

procedure

Defines the handling routines for clicking the mouse button.

Parameters:

max_clicks An integer expression representing the maximum number of mouse clicks to be counted.

butn_mask An integer expression representing the button mask the program is waiting for.

- 1 Left Button
- 2 Right Button
- 3 Both Buttons

butn_status An integer expression representing the button status the program is waiting for.

- 1 Left Button pressed.
- 2 Right Button pressed.
- 3 Both Buttons pressed.

The button mask and button status may be combined in different ways to obtain results:

<u>Button Mask</u>	<u>Button Status</u>	<u>Result</u>
1	1	Wait for left button press
1	3	Wait for left button press
1	0	Wait for left button up
1	2	Wait for left button up
2	2	Wait for right button press
2	3	Wait for right button press
2	0	Wait for right button up
2	1	Wait for right button up
3	1	Wait for left button press
3	2	Wait for right button press
3	3	Wait for both buttons pressed
3	0	Wait for both buttons up

If more than one button status event must be checked for, use two or more calls to this function, and alternate between them.

Example:

```
ON MENU BUTTON 4,1,1 GOSUB button_procedure
DO
'This loop does nothing but wait
LOOP
'
PROCEDURE button_procedure
PRINT "You pressed the Left Button ";MENU(15);" times."
RETURN
```

Related Words:

ON MENU KEY, ON MENU, ON MENU MESSAGE, ON MENU BUTTON

ON...GOSUB**Statement**

Syntax: **ON expression GOSUB proc_list**

Allows redirection of a program to one of a list of procedures.

Parameters:

expression Any numeric expression

proc_list A group of procedure names, separated by commas. If the value of the integer expression is greater than or equal to one, the first procedure on the list is executed. If the value is two, the second procedure is executed and so forth. No procedure is called if the number falls outside of the number of procedures listed.

Example:

```
DO  
  INPUT "Press a number key";a  
  ON a GOSUB proc_1,proc_2  
LOOP
```

```
'  
PROCEDURE proc_1  
  PRINT "You pressed 1"  
RETURN
```

```
'  
PROCEDURE proc_2  
  PRINT "You pressed 2"  
RETURN
```

Only pressing the number "1" or "2" keys will cause a response from the program.

**OPEN
Statement****Syntax:****OPEN "mode",[#]num,"filename"[,len]****Abbreviation:****O**

Opens a data channel to a disk file. A file may be opened twice, on different channels, and even with different modes.

Parameters:

mode Mode under which file is to opened. Must always be enclosed in quotes.

<u>Character</u>	<u>Mode</u>
O	Output
I	Input
A	Append to existing file
U	Enables both reading and writing to a file. The file must have previously been created by opening with "O".
R	Random Access file

num An integer expression between 0 and 99 defining the channel number which will be used for input and output.

filename The name of the file. The hierarchical file system may be used.

[,len] Used only in Random access files. This is the length of a file entry. If not present, a length of 128 bytes is designated.

The access codes (O, I, A, U, R) are not required in all cases. An empty (null) string ("") may be inserted, and the file may be directed to one of the following devices:

<u>Device Name</u>	<u>Device</u>
CON:	Console
LST:	Printer
PRN:	Printer
AUX:	Serial interface cards
VID:	Console (transparent mode) Commands are shown on the screen as special characters, not executed.
IKB:	Direct access to the 6301 Keyboard processor.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT #1,"Test line"
CLOSE #1
OPEN "I",#2,"TEST.DAT"
DO
    EXIT IF EOF(#2)
    PRINT INPUT$(1,#2)
LOOP
CLOSE
```

Related Words:
CLOSE, INPUT#, PRINT#

OPENW
Statement

Syntax: **OPENW** *num* [*x,y*]

Abbreviation: **O W**

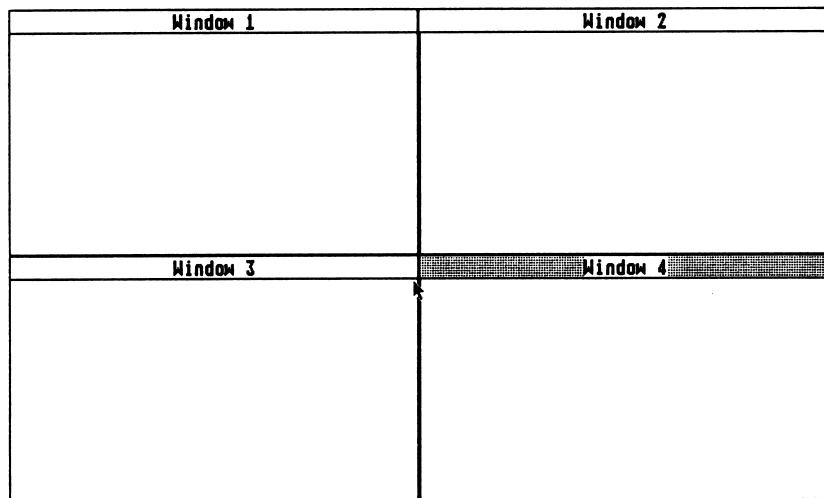
Opens a window on the screen.

Parameters:

num A numeric expression, between 0 and 4, representing the number of the window to open.

x An optional numeric expression representing the X-coordinate of the point of contact of the four possible windows on the screen. (See Figure 2-6.)

y An optional numeric expression representing the Y-coordinate of the point of contact of the four possible windows on the screen. (See Figure 2-6.)

Figure 2-6 Four possible window locations

Using the statement OPENW 0 opens the whole screen, without the menu bar as a window. The starting point for graphics commands is set with OPENW 0,x,y.

The x and y coordinates should only be set for one window. If it's necessary to change the coordinates, it's best to close all windows and reopen them.

Example:

OPENW 2,320,200

Related Words:
TITLEW, FULLW

OPTION
Statement

Syntax: **OPTION ["text"]**

Abbreviation: OPT

Changes the control for the compiler.

Parameters:

text Allows control commands to be passed to the compiler. (See Chapter 9, Compiling GFA BASIC programs.)

OPTION BASE**Statement****Syntax:****OPTION BASE** *num***Abbreviation:****OPT**

Changes the array base number.

Parameters:

num 0 or 1, setting the lower limit for array dimensioning. OPTION BASE 0 allows array(0) to be addressed. OPTION BASE 1 sets the lower limit to array(1).

Example:

```
OPTION BASE 1
DIM a(100)
FOR ctr%=1 TO 100
  a(ctr%)=5
NEXT ctr%
```

OR
Conditional Operator

Syntax: *condition OR condition*

OR is a conditional operator, usually used with an IF...THEN, REPEAT...UNTIL, or WHILE...WEND statements. OR does not evaluate conditions, but logically compares the results.

Parameters:

condition Any conditional statement, such as a logical expression, arithmetic equation, or comparison.

<u>Condition A</u>	<u>OR</u>	<u>Condition B</u>	<u>Result</u>
-1 (true)	OR	-1 (true)	-1 (true)
-1 (true)	OR	0 (false)	-1 (true)
0 (false)	OR	-1 (true)	-1 (true)
0 (false)	OR	0 (false)	0 (false)

Example:

```
a=10  
b=15  
IF a=10 OR b=13  
    PRINT "At least one condition was true"  
ENDIF
```

Related Words:

AND, XOR, NOT, IF, WHILE, REPEAT

OUT
Statement

Syntax: OUT *port,byte*
 OUT #*filenumber,byte*

Transfers a byte of data to a peripheral device, or a disk file.

Abbreviation: OU

Parameters:

port An integer expression representing a peripheral device.

<i>Number</i>	Device	
0	LST:	(Printer Port)
1	AUX:	(Serial Port RS232)
2	CON:	(Screen)
3	MID:	(MIDI Port)
4	IKB:	(Intelligent Keyboard Device.) Use caution here! Spectacular crashes can occur.
5	VID:	(Screen)

#filenumber An integer number between 0 and 99, representing the channel number assigned to the file by using the OPEN statement.

byte An integer expression containing one byte of data to be transferred to a peripheral device or disk file.

Example:

```
OPEN "O",#1,"TEST.DAT"
OUT #1,65
OUT #1,66
CLOSE #1
OPEN "I",#1,"TEST.DAT"
PRINT INPUT$(2,31)
CLOSE #1
OUT 2,67
```

Related Words:
INP, PRINT, PRINT#, INPUT, INKEY\$, INPUT#

OUT?
Function**Syntax:** *integer = OUT?(port)*

Determines the status of an output device. A zero (FALSE) is returned if the device is not ready to receive a character. The value -1 (TRUE) is returned when the device is ready to receive a character.

Parameters:*integer* Any integer expression.*port* An integer expression representing the number of a peripheral device.

<u>Number</u>	<u>Device</u>
0	LST: (Printer Port)
1	AUX: (Serial Port RS232)
2	CON: (Screen)
3	MID: (MIDI Port)
4	IKB: (Intelligent Keyboard Device.) Use caution here! Spectacular crashes can occur.
5	VID: (Screen)

Example:

```
a% = OUT?(0)
IF a% <> 0
  PRINT "Printer not ready"
ELSE
  PRINT "Printer ready"
ENDIF
```

Related Words:
INP?

PAUSE
Statement**Syntax:** PAUSE *integer_expression***Abbreviation:** PA

Stops program execution for an exactly defined period of time.

Parameters:integer_expression* An integer expression, between -2147483648 and +2147483647, defining the length of time the program should wait before continuing.

The length of the pause may be calculated as:

$$X = \text{delay(in secs.)} * 50$$

Example:

```
PRINT "Waiting for 10 seconds to pass...."  
PAUSE 500  
PRINT "Time's up"
```

PBOX
Function**Syntax:** PBOX *x0,y0,x1,y1***Abbreviation:** PB

This function draws a solid rectangular shape by specifying the diagonally opposed corners. The color of the rectangle is set by using the COLOR function. The style of the fill may be specified by using DEFFILL.

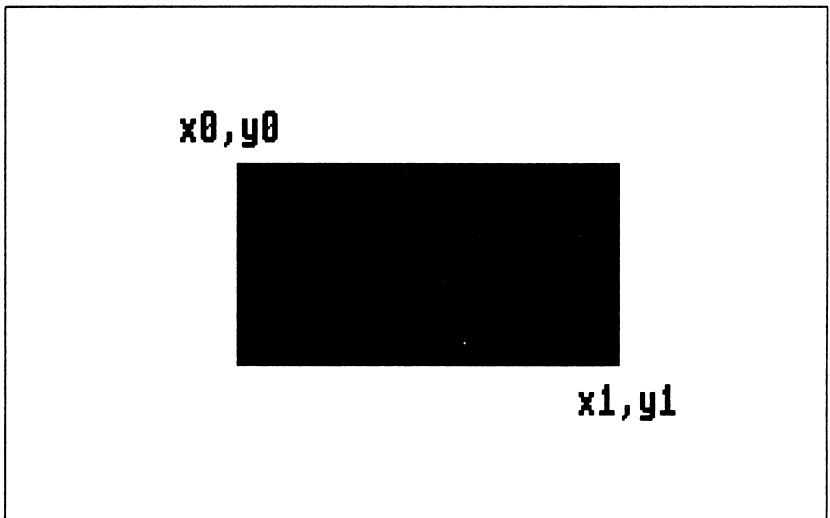
x0 X coordinate of upper left hand corner of rectangle.

y0 Y coordinate of upper left hand corner of rectangle.

x1 X coordinate of lower right hand corner of rectangle.

y1 Y coordinate of lower right hand corner of rectangle.

The coordinates are not required to be on the screen area, in which case only the portion of the rectangle which appears on the screen will be displayed.



Example:

```
CLS  
PBOX 100,100,150,150  
PBOX 175,180,300,400
```

Related Words:
BOX, PRBOX, RBOX

PCIRCLE**Statement****Syntax:** PCIRCLE *x,y,r [,arc1,arc2]***Abbreviation:** PC

Draws a filled (solid) circle, or pie slice (arc). The type of fill pattern and color must be selected with DEFFILL.

x X coordinate of center of circle or arc.

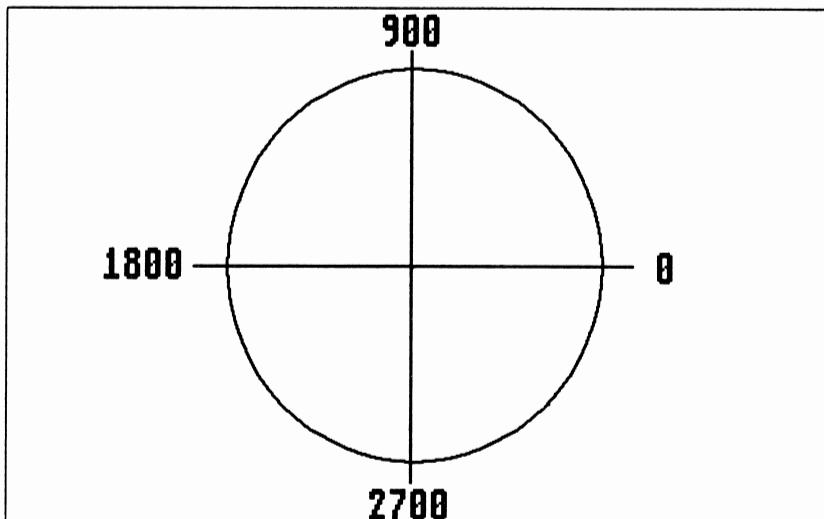
y Y coordinate of center of circle or arc.

r Radius (in pixels) of circle or arc.

arc1 Beginning angle of arc to be drawn. In tenths of a degree (0-3600).

arc2 Ending angle of arc to be drawn. In tenths of a degree (0-3600).

0	Right
900	Top
1800	Left
2700	Bottom

Figure 2-7 Circle**Example:**

```
PCIRCLE 100,100,20  
PCIRCLE 150,150,20,900,1300
```

This example draws a complete filled circle, centered at 100,100, with a radius of 20, and a pie-slice centered at 150,150, with a radius of 20, and a starting angle of 90 degrees (at the top) ending at 130 degrees.

PEEK
Function

Syntax: **PEEK address**

Returns the contents of one byte of memory.

Parameters:

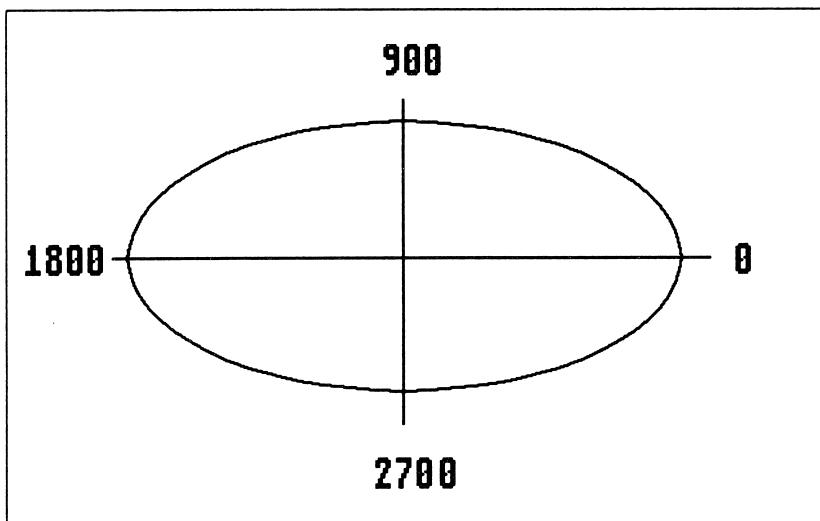
address A numeric expression which represents a location in memory to be examined.

Example:

```
a$="A"
desc=ARRPTR(a$)
addr=LPEEK(d)
length=DPEEK(d+4)
ascii=PEEK(a)
PRINT "Descriptor Address: ";desc
PRINT "Address          : ";addr
PRINT "Length           : ";length
PRINT "ASCII Value     : ";ascii
```

Related Words:
PEEK, LPEEK, DPEEK**PELLIPSE****Statement****Syntax:** PELLIPSE x,y,r [,arc1,arc2]**Abbreviation:** PE**Draws a filled ellipse, or part of an ellipse (arc).****Parameters:****x** X coordinate of center of ellipse or arc.**y** Y coordinate of center of ellipse or arc.**r** Radius (in pixels) of ellipse or arc.**arc1** Beginning angle of arc to be drawn. In tenths of a degree (0-3600).**arc2** Ending angle of arc to be drawn. In tenths of a degree (0-3600).

0	Right
900	Top
1800	Left
2700	Bottom

Figure 2-8 Pellipse**Example:**

```
PELLIPSE 100,100,20  
PELLIPSE 150,150,20,900,1300
```

This example draws a solid ellipse, centered at 100,100, with a radius of 20, and an arc centered at 150,150, with a radius of 20, and a starting angle of 90 degrees (at the top) ending at 130 degrees.

Related Words:
CIRCLE, ELLIPSE, PCIRCLE

PI
Function**Syntax:** **PI**

Returns the value of the function, PI (3.141592653. . .).

Example:

```
INPUT "Radius of circle";r
c=2*PI*r
a=PI*r*r
PRINT "Circumference of circle: ";c
PRINT "Area of circle      : ";a
```

PLOT
Statement**Syntax:** **PLOT x,y****Abbreviation:** **PL**

Draws one point on the screen. The DEFLINE function may be used to define the style to be used for plotting the point on the screen.

Parameters:

x X coordinate of the point.

y Y coordinate of the point.

Example:

```
a%=125
b%=150
PLOT 10,10
PLOT 100,150
PLOT a%,b%
```

POINT
Function**Syntax:** **POINT(x,y)**

Checks a pixel on the screen and returns the value of the color register of that pixel. On a high resolution (monochrome) screen, the values zero or one are returned. Medium resolution (color) screens will return a value of zero through three. Low resolution (color) screens will return a value of zero through 15.

Parameters:

- x** X coordinate of point.
- y** Y coordinate of point.

Example:

```
PLOT 100,100
PRINT POINT(99,100)
PRINT POINT(100,100)
a%=100
b%=100
PRINT POINT(a%,b%)
```

POKE
Function**Syntax:** **POKE address,number****Abbreviation:** **PO**

Stores 1 byte into an area of main memory at a specified address.

Parameters:

- address** A numeric expression which specifies an even address in memory.
- number** A numeric expression between 0 and 255.

Example:

```
a$="A"
ptr=ARRPTR(a$)
DPOKE ptr+4,4
z=VARPTR(a$)
LPOKE z,111638594
PRINT a$
POKE z,67
PRINT a$
```

Related Words:

DPOKE, LPOKE, SPOKE, SLPOKE

POLYFILL

Function

Syntax: **POLYFILL** *number,array_1(),array_2()* [OFFSET *x0,y0*]

Abbreviation: **POLYF**

Draws a filled in shape. The first and last dot are joined by a straight line, and the space inside the shape is filled with a pattern selected by DEFFILL. Areas where lines cross are not filled, however. Note that the arrays are passed without any index numbers.

Parameters:

number A numeric expression, with a maximum value of 128, representing the number of points which are joined by straight lines. If a closed shape is defined, the first and last points must be the same.

array_1() Numeric field which contain the X coordinate points for the shape to be drawn. The X coordinate of the first point must be stored in *array_1(0)*.

array_2() Numeric field which contain the Y coordinate points for the shape to be drawn. The Y coordinate of the first point must be stored in *array_2(0)*.

x0 An offset defined to display the shape offset horizontally by a specified amount.

y0 An offset defined to display the shape offset vertically by a specified amount.

Example:

```
DIM x(10),y(10)
DATA 10,10,20,30,5,45,100,100,200,10,10,10
FOR ctr%=0 TO 5
    READ x(ctr%),y(ctr%)
NEXT ctr%
t=TIMER
PRINT "Standby for POLYFILL..."
PAUSE 100
CLS
PRINT AT(1,23); "Defined shape with POLYFILL ";
t=TIMER
POLYFILL 6,x(),y()
PRINT (TIMER-t)/200; " Secs."
PAUSE 250
```

Related Words:

POLYMARK, POLYLINE

POLYLINE

Function

Syntax:**POLYLINE** *number,array_1(),array_2() [OFFSET x0,y0]***Abbreviation:****POL**

Draws a shape. The first and last dot are joined by a straight line. This function is similar to DRAW, but much faster. Note that the arrays are passed without any index numbers.

Parameters:

number A numeric expression, with a maximum value of 128, representing the number of points which are joined by straight lines. If a closed shape is defined, the first and last points must be the same.

array_1() Numeric field which contain the X coordinate points for the shape to be drawn. The X coordinate of the first point must be stored in *array_1(0)*.

array_2() Numeric field which contain the Y coordinate points for the shape to be drawn. The Y coordinate of the first point must be stored in *array_2(0)*.

x0 An offset defined to display the shape offset horizontally by a specified amount.

y0 An offset defined to display the shape offset vertically by a specified amount.

Example:

```
DIM x(10),y(10)
DATA 10,10,20,30,5,45,100,100,200,10,10,10
FOR ctr%=0 TO 5
  READ x(ctr%),y(ctr%)
NEXT ctr%
t=TIMER
PRINT "Standby for POLYLINE..."
PAUSE 100
CLS
PRINT AT(1,23);"Defined shape with POLYLINE";
t=TIMER
POLYLINE 6,x(),y()
PRINT (TIMER-t)/200;" Secs."
PAUSE 250
```

Related Words:
POLYMARK, POLYFILL

POLYMARK

Function

Syntax: **POLYMARK** *number,array_1(),array_2()* [OFFSET *x0,y0*]

Abbreviation: **POL**

Marks the corner points of a shape with symbols as defined by the function DEFMARK. Note that the arrays are passed without any index numbers.

Parameters:

number A numeric expression, with a maximum value of 128, representing the number of points which are joined by straight lines. If a closed shape is defined, the first and last points must be the same.

array_1() Numeric field which contains the X coordinate points for the shape to be drawn. The X coordinate of the first point must be stored in *array_1(0)*.

array_2() Numeric field which contains the Y coordinate points for the shape to be drawn. The Y coordinate of the first point must be stored in *array_2(0)*.

x0 An offset defined to display the shape offset horizontally by a specified amount.

y0 An offset defined to display the shape offset vertically by a specified amount.

Example:

```
DIM x(10),y(10)
DATA 10,10,20,30,5,45,100,100,200,10,10,10
FOR ctr%=0 TO 5
    READ x(ctr%),y(ctr%)
NEXT ctr%
t=TIMER
PRINT "Standby for POLYMARK..."
PAUSE 100
CLS
PRINT AT(1,23); "Defined shape with POLYMARK ";
t=TIMER
DEFMARK 1,3
POLYMARK 6,x(),y()
PRINT (TIMER-t)/200; " Secs."
PAUSE 250
```

Related Words:

POLYFILL, POLYLINE

POS

Function

Syntax: **POS(*number*)**

Returns the column in which the cursor is located. May not correspond to the actual cursor location after using PRINT AT, cursor commands, etc., which move the cursor.

Carriage return, CHR\$(13), line feed, CHR\$(10), and backspace, CHR\$(8), reset the value of POS.

If a string longer than 80 characters is displayed, the value of POS will also be greater than 80.

Parameters:

number A dummy value. Any integer value may be used. It has no effect on the function.

Example:

```
FOR ctr%-1 TO 600
    PRINT "A";
    IF POS(1)=30
        PRINT
    ENDIF
NEXT ctr%
```

PRBOX**Function****Syntax:** PRBOX x0,y0,x1,y1**Abbreviation:** PRB

This function draws a solid rectangular shape with rounded corners by specifying the diagonally opposed corners. The color of the rectangle is set by using the COLOR function. The fill style may be specified by DEFILL.

x0 X coordinate of upper left hand corner of rectangle.

y0 Y coordinate of upper left hand corner of rectangle.

x1 X coordinate of lower right hand corner of rectangle.

y1 Y coordinate of lower right hand corner of rectangle.

The coordinates are not required to be on the screen area, in which case only the portion of the rectangle which appears on the screen will be displayed.

Example:

```
CLS
PRBOX 100,100,150,150
PRBOX 175,180,300,400
```

Related Words:

BOX, PBOX, RBOX

PRINT
Statement

Syntax: **PRINT "text"**
 PRINT variable

Abbreviation: P or ?

Displays information on the screen.

Parameters:

text A character expression.

variable Any legal variable name. A group of variables may be listed. If the variables in a list are separated by a semicolon (;), they are displayed one after the other. If the expressions are separated by commas (,), they are displayed at intervals of 16 columns. If an apostrophe is placed between the expressions, one space is left for each apostrophe.

If the command ends with a semicolon, comma or apostrophe, the carriage return/line feed is suppressed.

Example:

```
PRINT "Hello"  
a=1  
PRINT a  
PRINT "Hello","Again";a,a
```

Related Words:

PRINT AT, PRINT #, PRINT USING

PRINT AT
Statement**Syntax:**

PRINT AT(x,y);"text"
PRINT AT(x,y); variable

Abbreviation:

P AT

Displays information at a specified location on the screen.

Parameters:

text A character expression.

variable Any legal variable name. A group of variables may be listed. If the variables in a list are separated by a semicolon (;), they are displayed one after the other. If the expressions are separated by commas (,), they are displayed at intervals of 16 columns. If an apostrophe is placed between the expressions, one space is left for each apostrophe.

If the command ends with a semicolon, comma or apostrophe, the carriage return/line feed is suppressed.

x Cursor column for displaying text.

y Cursor row for displaying text.

Example:

```
PRINT AT(10,10);"Hello"
a=1
b$="Hello again."
PRINT AT(15,15); a
PRINT AT(20,10);b$
```

Related Words:

PRINT, PRINT #, PRINT USING

PRINT #
Statement

Syntax: **PRINT #channel, "text"**
 PRINT #channel, variable

Abbreviation: P

Prints the information contained in an expression to a data channel.

Parameters:

channel A numeric expression between 0 and 99 corresponding to the channel number assigned by the OPEN statement.

text A character expression.

variable Any legal variable name. A group of variables may be listed. If the variables in a list are separated by a semicolon (;), they are printed one after the other. If the expressions are separated by commas (,), they are printed at intervals of 16 columns. An apostrophe is placed between the expressions, one space is left for each apostrophe.

If the command ends with a semicolon, comma or apostrophe, the carriage return/line feed is suppressed.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT #1,"Hello"
a=1
b$="Hello again."
PRINT #1,a
PRINT #1,b$
CLOSE #1
```

Related Words:
PRINT AT, PRINT USING, PRINT

PRINT USING**Statement****Syntax:****PRINT USING "format",exp_list[;]****Abbreviation:****P USING**

This is a very powerful, and very complex form of the PRINT statement. It is used to precisely format printed output.

Parameters:

format This is a template which defines the format for displaying the text. See the accompanying table for the formatting instructions.

exp_list A list of expressions, separated by commas.

<u>Format</u>	<u>Description</u>
#	Reserves space for digits. If enough digits to fully display the number, it will be printed with a percent sign (%) at the start of the number, ignoring the formatting string.
.	Positions the decimal point.
+	Displays a leading plus sign.
-	Reserves space for a leading minus sign. The minus sign is only printed if the number is negative.
*	Reserves space for a digit. The asterisk is printed if no digit is printed in that space.
\$\$ or \$	Prints a leading dollar sign (\$).
,	Inserts a comma after every third digit in a number.
^^^^	Display in exponential form. The number of carats corresponds to the number of digits plus E and the sign (+/-).
!	Indicates that the first character of the string is to be printed.
&	The whole string is printed.

\.\

As many characters of the string are printed as there are characters in the template, including backslashes.

-

The underline character treats the next character in the format string as a character, rather than as a format character. Thus PRINT USING “_&&”,“TEST” displays “&TEST”.

PRINT #channel, USING is also permitted, and formats the digits and character strings to a data channel. The channel number must have been previously assigned with the OPEN statement.

Example:

```
PRINT USING "###.##";PI
PRINT USING "+##.##";99.9999
PRINT USING "$##.##";110.987
PRINT USING "*****";123
```

Related Words:

PRINT, PRINT AT, PRINT #

PROCEDURE**Statement****Syntax:****PROCEDURE name(parameter_list)****Abbreviation:****PRO**

Marks the beginning of a Procedure (subroutine). Every procedure must be entered at the beginning, and exited only through the RETURN at the end of the procedure. GOTO is permitted within a Procedure, but you may not jump out of a Procedure.

A procedure may call another procedure, and even call itself.

Parameters:

name A procedure name may begin with a digit. Since GFA BASIC does not use line numbers, every procedure must have a name to be called from within the main program. Procedure names also may contain letters, digits, underscore, and the slash symbol (/).

parameter_list Variable names, separated by commas. Values are passed to these variables by the GOSUB statement. These parameters are always local variables. To pass a value back to the main program, equate the variable to a global variable, or use pointers.

Example:

```
PRINT "This is the main routine"
GOSUB proc_1
    PRINT "Back in main"
GOSUB proc_2(1,3,5)

PROCEDURE proc_1
    PRINT "This is the 1st Procedure."
RETURN

PROCEDURE proc_2(a,b,c)
    PRINT "This is the 2nd Procedure."
    PRINT a
    PRINT b
    PRINT c
RETURN
```

Related Words:
GOSUB, RETURN, LOCAL

PSAVE
Command**Syntax:** **PSAVE "filename"****Abbreviation:** **PS**

Saves a program to a disk in a form which cannot be listed. Any PSAVE'd program will begin to run as soon as the LOAD command is executed.

After the program has run, the program remains in memory, and may be run again by entering *run* in the command window, or by selecting *Run* from the Edit window menu, however, no listing is displayed in the Edit window.

You may not add lines by typing commands into the edit window. An "Exit without a Loop" error will be generated. Select new from the Edit Menu, or type new into the command window to clear the unlistable program from memory.

The hierarchical file system is permitted with this command.

CAUTION! If you PSAVE a program with a file name that has already been assigned to another program, that program will be overwritten by the newly PSAVE'd program, without a warning.

Parameters:

filename Any valid file name. The hierarchical file system may be used. If quotes are not placed around the filename, they are assumed to be present.

Example:**PSAVE "TEST.PRG"****PSAVE "A:TEST.PRG"****Related Words:**
SAVE, LOAD

PTSIN
System Variable

Contains the address of the VDI (Virtual Device Interface) Input parameter block.

Related Words:
PTSOUT, INTIN, INTOUT, GINTIN, GINTOUT, GB:, CONTROL, ADDRIN, ADDROUT

PTSOUT**System Variable**

Contains the address of the VDI (Virtual Device Interface) Output parameter block.

Related Words:

PTSOUT, INTIN, INTOUT, GINTIN, GINTOUT, GB; CONTROL, ADDRIN, ADDRROUT

PUT**Statement****Syntax:**

PUT var_1,var_2,string_var[,mode]

Abbreviation:

PU

PUT is used to place a graphics image on the screen. PUT is the compliment of the GET statement.

Parameters:

var_1 X coordinate of the upper left corner of the graphics block to be placed on the screen.

var_2 Y coordinate of the upper left corner of the graphics block to be placed on the screen.

string_var A character string variable expression which contains the bit pattern for the graphics block to be placed on the screen.

mode An optional parameter that is a numeric expression between 0 and 15, representing the manner in which the block will be displayed. If the mode is not specified, mode 3, replace is assumed.

Mode	Action
0	delete (empty rectangle)
1	Source AND Destination
2	Source AND (NOT Destination)
3	Source (overwrite)
4	(NOT Source) AND Destination
5	Destination (do nothing)
6	Source XOR Destination
7	Source OR Destination
8	NOT (Source OR Destination)
9	NOT (Source or Destination)
10	NOT Destination (inverse)
11	Source OR (NOT Destination)

12	NOT Source (Inverse overwrite)
13	(NOT Source) OR Destination
14	NOT (Source AND Destination)
15	1 (solid filled rectangle)

The most commonly used modes are mode 3 and 6. Mode 3 replaces whatever is on the screen with the bitpattern contained in the string variable expression.

Mode 6 XOR's the bitpattern against the screen, producing a shadow image. XORing the data twice removes it from the screen, leaving the original image untouched. This is useful for animation over a detailed background.

Example:

```
FOR ctr% = 0 to 4
  CIRCLE 150,100,ctr%*30
NEXT ctr%
GET 0,0,320,199,a$
PUT 250,0,a$,3
```

Related Words:

GET

PUT# Statement

Syntax: **PUT#[number,int_expr]**

Abbreviation: PU

Writes a record to a random access file.

Parameters:

number An integer expression between 0 and 99 which refers to the number of the data channel created by the OPEN statement.

int_expr An expression between 1 and the number of records in the file. The maximum number of records permitted is 65535.

NOTE: If the record number is not specified, the next record in the file is written.

Example:

See the entry for FIELD for an example program.

Related Words: GET#, FIELD

QUIT
Statement**Syntax:** **QUIT****Abbreviation:** **Q**

Stops execution of a program and exits GFA BASIC to the GEM desktop.
This statement is identical to the **SYSTEM** statement.

Example:

```
PRINT "Enter Password:"  
INPUT answer$  
IF answer$<>"GFA"  
    QUIT  
ENDIF
```

Related Words:
END, SYSTEM

RANDOM
Function**Syntax:** **RANDOM(*seed*)**

This function returns an integer random number between 0 and the seed value.

Parameters:

seed Any positive or negative integer value.

Example:

```
a=%RANDOM(6)
```

This example returns a random number between 0 and 6. The following example shows the use of RANDOM within a program. In this case, a random screen location is plotted, with a random color. The color is selected as either red, green, or blue. This example also demonstrates the degree of randomness of the RANDOM function by the distribution of the plotted pixels on the screen.

```
SETCOLOR 0,0,0
SETCOLOR 1,7,0,0
SETCOLOR 2,0,7,0
SETCOLOR 3,0,0,7
CLS
DO
    EXIT IF a$=CHR$(13)
    a$=INKEY$
    x%=RANDOM(640)
    y%=RANDOM(200)
    COLOR RANDOM(4+1)
    PLOT x%,y%
LOOP
SETCOLOR 0,7,7,7
SETCOLOR 3,0,0,0
```

Related Words:
RND

RBOX**Function****Syntax:****RBOX $x0,y0,x1,y1$** **Abbreviation:****RB**

This function draws a hollow rectangular shape with rounded corners, by specifying the diagonally opposed corners.

Parameters:

$x0$ X coordinate of upper left corner of box.

$y0$ Y coordinate of upper left corner of box.

$x1$ X coordinate of lower right corner of box.

$y1$ Y coordinate of lower right corner of box.

Example:**RBOX 10,10,100,100****Related Words:**

BOX, DEFLINE

READ
Statement**Syntax:** **READ** *variable_list***Abbreviation:** **REA**

Reads information from a DATA statement and stores the information in a variable or list of variables. Whenever the READ statement is executed, the next item pointed to by the data pointer is read into the variable or variable list. When the last item is read, the data pointer must be reset before the data can be read again.

The data obtained by the READ statement must correspond to the variable type specified.

The number of items in the DATA line must be equal to or greater than the number of variables in the READ statement. If the number of data items is greater, the excess items are ignored.

Parameters:

variable_list Any boolean, numeric, or character string variable. The variable type must be of the same type as the data item read.

Example:

```
READ a,b,c$  
PRINT a  
PRINT b  
PRINT c$  
READ d$  
PRINT d$  
  
DATA 1,12345,Hello  
DATA "This is GFA BASIC"
```

Related Words:
DATA, RESTORE

**RELSEEK
Statement**

Syntax: RELSEEK [#]number,int_expr

Abbreviation: REL

Positions the file pointer a specified number of bytes in a designated file.
The file pointer points to a specific byte in a file, and is found on every data channel.

Parameters:

number The channel number assigned by the OPEN statement.

int_expr The number of bytes into the file to move the file pointer. This is an integer expression within the limits of the file.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT#1,"1234567890"
CLOSE#1
OPEN "R",#1,"TEST.DAT"
RELSEEK#1,5
INP#1,a
PRINT a
CLOSE#1
```

Related Words:
LOC, SEEK

REM
Statement**Syntax:** **REM** *text***Abbreviation:** R or ' or !

This statement allows notes to be included within the program. This statement is not executed. Everything in a line following a REM statement is ignored by the interpreter or compiler.

The apostrophe (') may be used at the beginning of a line instead of the REM statement. This is not an abbreviation, but is accepted as a symbol for the REM statement.

To attach a comment to the end of a line of code, use the exclamation point (!).

Parameters:

text Any note or comment that is to be included in the source code, but not executed.

Example:

```
REM Sample Program  
' January 1, 1988  
PRINT "Hello" !This is a remark, too.
```

REPEAT...UNTIL
Statement**Syntax:** **REPEAT**
 program_statements
 UNTIL *condition***Abbreviation:** **REPEAT**
 U

Creates a looping structure which is executed until a predefined condition is true. The condition for ending the loop is checked at the end of the loop.

Parameters:

program_statements A statement or group of statements to be executed continuously until the end condition is satisfied.

condition A condition that is checked to determine if the loop should end. The loop is repeated if the condition is not evaluated as true.

Example:

```
REPEAT
    PRINT a%
    INC a%
UNTIL a%=10
```

Related Words:

WHILE...WEND, IF...THEN

RESERVE

Statement

Syntax:

RESERVE *numeric_expr*

Abbreviation:

RESE

Increases or decreases the amount of memory available to be used by GFA BASIC. RESERVE allows areas of memory to be protected from GFA BASIC for use with machine language routines or .RSC files. Remember to always restore the memory segment reserved before exiting a program.

Parameters:

numeric_expr A number which determines the amount of memory available after this statement is executed.

Example:

```
PRINT FRE(0)
RESERVE FRE(0)-32767
PRINT FRE(0)
RESERVE FRE(0)+32767
```

Related Word:

FRE()

RESTORE**Function****Syntax:** **RESTORE [*label*]****Abbreviation:** **RES**

Resets the data pointer to the first item of data in a program, or to the first item of data following a specified label. This function permits repeated access to the same data lines.

Parameters:***label*** Any legal label.**Example:**

```
DO
  READ a%,a$
  PRINT a%,a$
  IF a%-3
    RESTORE
  ENDIF
LOOP
```

```
DATA 1,item 1
DATA 2,item 2
DATA 3,item 3
```

This example creates an endless loop, reading the same data again and again. Press the <Control>, <Shift>, <Alternate> keys to exit.

Related Words:**DATA, READ**

RESUME
Statement**Syntax:****RESUME**
RESUME NEXT
RESUME *label***Abbreviation:****RESU**

Defines the method for exiting an error handling routine.

RESUME repeats the command which generated an error.

RESUME NEXT continues program execution with the statement following the error generating statement.

RESUME *label* causes the program to branch to a routine marked by a specified label. If the label is in the main program, the **RETURN** at the end of the error procedure is canceled, and all global variables are restored.

In the event of a fatal error, only **RESUME *label*** is possible.

Parameters:

label Any legal label name.

Example:

```
ON ERROR GOSUB error_handler
PRINT SQR(1)
PRINT 3/0
ON ERROR
PRINT SQR(1)

PROCEDURE error_handler
PRINT "Error number: ";ERR
ON ERROR GOSUB error_handler
RESUME NEXT
RETURN
```

Related Words:

ERR, ON ERROR

RETURN
Statement**Syntax:** **RETURN****Abbreviation:** **RET**

This statement ends a procedure and resumes program execution with the program statement immediately following the GOSUB statement which called the procedure, and any local variables defined by the procedure are deleted.

Every procedure must end with a RETURN statement.

Example:

```
PRINT "This is the Main program"
GOSUB proc_1
a%=5
b%=6
c$="Hello"
PRINT "Back in the Main program"
PRINT a%,b%,c$
'

PROCEDURE proc_1
LOCAL a%,b%,c$
a%=1
b%=2
c$="GFA BASIC"
PRINT "This is the Subroutine"
PRINT a%,b%,c$
RETURN
```

Related Words:
GOSUB, @

RIGHT\$
Function

Syntax: **RIGHT\$(*string_var[,int]*)**

Extracts a specified number of characters, or if unspecified, the last character, from a character string or string expression, beginning with the rightmost character.

Parameters:

string_var Any character string or string variable.

int A numeric expression which specifies the number of characters to be read from the rightmost character of the string. Integer or real numbers are permitted, however, in the case of a real number, only the integer portion will be considered.

If this number is greater than the length of the string, the whole string will be returned.

If equal to zero, an empty string will be returned.

Example:

```
s$="GFA BASIC"  
b$=RIGHT$(s$)  
c$=RIGHT$(s$,5)  
PRINT s$  
PRINT b$  
PRINT c$
```

Related Words:
LEFT\$, MID\$

RMDIR
Command**Syntax:** **RMDIR " pathname "****Abbreviation:** **RM**

This command is used to delete or remove a folder from a disk. The hierarchical file system may be used to specify the path for deleting a folder.

Parameters:

pathname The name of the folder to be removed from the disk.

Example:

```
MKDIR "A:\SAMPLE"  
FILES "A:\*.*"  
RMDIR "A:\SAMPLE"  
FILES "A:\*.*"
```

Related Words:
CHDIR, MKDIR, KILL

RND
Function**Syntax:** **RND[(seed)]**

Returns a real random number between 0 and 1.

Parameters:

seed An optional dummy parameter, which is disregarded.

Example:

```
FOR ctr%=1 TO 10  
    PRINT RND  
NEXT ctr%  
FOR ctr%=1 TO 10  
    PRINT INT(RND*10)+1  
NEXT ctr%
```

The second part of this example generates random numbers between 1 and 10.

Related Words:
RANDOM

RSET
Function

Syntax: **RSET variable = expression**

Abbreviation: **RS**

Right justifies a string variable. Normally used in conjunction with FIELD when creating a Random access file.

Parameters:

variable Any legal string variable name. This variable should be previously defined with a template containing the maximum number of characters permitted in the string to be justified.

expression Any string expression. In the case of numeric values, the functions MKI\$, MKS\$, MKL\$, or MKD\$ must be used to transform the data into a string variable before calling this function. If the total number of characters in this expression is greater than the maximum allowable number of characters, the string is truncated. If the string is shorter, then the unused characters will contain spaces.

Example:

```
a$="AAAAAAAAA"  
b$=SPACE$(9)  
c$="Sample"  
RSET a$=c$  
RSET b$=c$  
PRINT c$  
PRINT a$  
PRINT b$  
RSET b$="This string is truncated"  
PRINT b$
```

Related Words:
LSET

RUN**Command****Syntax:** **RUN**

This command instructs GFA BASIC to begin execution of a program in memory. All previously defined variables are deleted.

Example:

```
DIM a$(20)
FOR ctr%=0 TO 20
    A$(ctr%)=CHR$(65)+ctr%
NEXT ctr%
PRINT "Program is running..."
FOR ctr%=0 TO 20
    PRINT a$(ctr%)
NEXT ctr%
PRINT "Ending program..."
RUN
```

Note: This program is an endless loop. It will execute, then run itself again. To exit, press <Control>, <Shift>, <Alternate>.

SAVE**Command****Syntax:** **SAVE "filename"****Abbreviation:** **SA**

Saves a program to a disk. If a file with the specified name is present, that file will be renamed to *filename*.BAK.

CAUTION! If you SAVE a program with a file name that has already been assigned to another program, that program will be overwritten by the newly SAVE'd program without warning.

Parameters:

filename Any valid file name. The hierarchical file system may be used. If quotes are not placed around the filename, they are assumed to be present.

Example:

```
SAVE "TEST.PRG"  
SAVE "A:\TEST.PRG"
```

Related Words:
PSAVE, LOAD

SDPOKE**Function**

Syntax: **SDPOKE** *addr,num_expr*

Abbreviation: **SD**

Stores 2 bytes in a designated area of main memory. This function operates in the Supervisor mode of the 68000 CPU, so reserved memory may be addressed. Otherwise, this command is identical to POKÉ.

Parameters:

addr The address in main memory for storing the number. The address must be an even number.

num_expr A number between 0 and 65535 to be placed in memory.

Example:

```
a$="A"  
a%=VARPTR(a$)  
PEEK a%  
SDPOKE a%,66  
PEEK a%  
PRINT a$
```

Related Words:
POKE, DPOKE, LPOKE, SPOKE, SLPOKE

SEEK
Function**Syntax:** **SEEK[#]file_number,byte****Abbreviation:** SEE

Sets the file pointer at a specified byte within a designated file.

Parameters:

file_number A number between 0 and 99 designating the file number, as assigned by the OPEN statement.

byte An integer expression representing the number of bytes within the file to position the file pointer. This number must be less than or equal to the length of the file.

Example:

```
OPEN "O",#1,"TEST.DAT"
PRINT#1,"ABCDEFGHIJKL"
SEEK#1,5
PRINT LOC(#1)
CLOSE#1
```

Related Words:

OPEN, CLOSE, PRINT#, WRITE#, LOC, LOF

SETCOLOR
Function**Syntax:** **SETCOLOR register,red,green,blue****SETCOLOR register,palette****Abbreviation:** SE

Permits the red, green and blue components of a specified color register to be defined. This function is similar to XBIOS 7, the setcolor XBIOS function.

Parameters:

register A number representing the color register to be changed. This number is dependent on the screen resolution. Low resolution (color) screens may have a register number between 0 and 15. Medium resolution (color) screens have register numbers between 0 and 3. High resolution (monochrome) screens only have registers 0 and 1. High resolution screens may only be set as:

SETCOLOR 0,0

or

SETCOLOR 0,1

red A number between 0 and 7 representing the amount of red to be mixed into the color palette.

green A number between 0 and 7 representing the amount of green to be mixed into the color palette.

blue A number between 0 and 7 representing the amount of blue to be mixed into the color palette.

palette An optional method to pass the value of the color register is to define a number calculated as:

number=red*256+green*16+blue

The higher the value of a red, green or blue component of a color, the more of that color will show. SETCOLOR 0,7,0,0 will set color register 0, normally the background color, to red on a color monitor, while SETCOLOR 0,0,0,7 will set the background color to blue.

Example:

```
SETCOLOR 0,0,0,0
SETCOLOR 1,7,7,7
PAUSE 50
SETCOLOR 0,0,0,7
SETCOLOR 1,0,7,0
```

SETTIME**Function**

Syntax: **SETTIME** *time_string,date_string*

Abbreviation: **SETT**

This function sets the ST's internal clock. This function is similar to XBIOS \$16, the set time/date function of XBIOS. If you wish to alter only the date or time, enter an empty string for the parameter to remain unchanged.

Parameters:

time_string A string expression containing the time. Hours, minutes and seconds may be entered. Two digits must be entered for each number. For instance, 1 AM is entered as 01. Hours must be entered in 24 hour (military) format. 1 PM must be entered as 13. The correct string for entering 4 PM is 16:00:00.

date_string A string expression containing the month, day, and year. Two digits must be used to express the month and day. The first two digits of the year are optional when entering dates between 1980 and 2079. The month, day, and year must be separated by a slash.

Example:

```
PRINT "Clock is set to: ";TIME$,DATE$  
INPUT "What is the correct time",t$  
INPUT "What is the date",d$  
SETTIME t$,d$  
PRINT TIME$,DATE$
```

Note that TIME\$ and DATE\$ are reserved words and may not be used as variables.

Related Words:

TIME\$, DATE\$

SGET
Function**Syntax:** **SGET** *string_var***Abbreviation:** **SG**

Stores the entire screen (32,000 bytes) into a string variable. This function is extremely fast. SGET is faster than using GET for the entire screen.

Parameters:*string_var* Any legal string variable name.**Example:**

```
PRINT AT(10,10);"Screen 1"  
SGET screen_1$  
CLS  
PRINT AT(10,10);"Screen 2"  
SGET screen_2$  
CLS  
DO  
    SPUT screen_1$  
    PAUSE 50  
    SPUT screen_2$  
    PAUSE 50  
LOOP
```

Related Words:
SPUT, GET, PUT

SGN
Function**Syntax:** **SGN**(*number*)

This is the mathematical sign function and determines whether the sign of a number is positive, negative, or zero. If the number is positive, the value +1 is returned. A negative number returns the value -1. If the number is equal to zero, a value of zero is returned.

Parameters:*number* Any numeric expression.

Example:

```
DIM a%(5)
FOR ctr%=0 TO 5
  READ a%(ctr%)
NEXT ctr%
FOR ctr%=0 TO 5
  PRINT a%(ctr%),SGN(a%(ctr%))
NEXT ctr%
DATA 100,50,-330,0,-25
```

Related Words:

ABS

SHOWM

Function

Syntax: SHOWM

Abbreviation: SH

Causes the mouse pointer to appear, if it has been previously hidden, or if something has been placed on the screen causing the mouse pointer to disappear.

Example:

```
HIDEM
PRINT "Mouse pointer is hidden"
PAUSE 250
SHOWM
PRINT "Now Mouse pointer appears."
DO
  PLOT MOUSEX,MOUSEY
LOOP
```

Related Words:

HIDEM

SIN
Function**Syntax:** **SIN(angle)**

This function computes the trigonometric SINE function of an angle (expressed in radians). One radian equals approximately 57 degrees.

Parameters:

angle A numeric expression which specifies the angle, in radians, for which the SIN function is to be calculated. If the desired angle is expressed in degrees, multiply the angle by PI/180.

Example:

```
INPUT "Enter the angle (in Radians)";angle  
PRINT "SIN= ",SIN(angle)
```

Related Words:

COS, TAN

SLPOKE
Function**Syntax:** **SLPOKE addr,num_expr****Abbreviation:** **SL**

Stores 4 bytes in a designated area of main memory. This function operates in the Supervisor mode of the 68000 CPU, so reserved memory may be addressed. Otherwise, this command is identical to POKE.

Parameters:

addr The address in main memory for storing the number. The address must be an even number.

num_expr A number between -2147483648 and +2147483647 to be placed in memory.

Example:

```
a$="A"
a%=VARPTR(a$)
LPEEK a%
SLPOKE a%,1111638594
LPEEK a%
PRINT a$
```

Related Words:

POKE, DPOKE, LPOKE, SPOKE, SDPOKE

SOUND
Function**Syntax:**

SOUND *channel, volume, note, octave [,duration]*

SOUND *channel, volume, #period [,duration]*

Abbreviation:

SO

This function generates sounds utilizing the ST's sound chip.

Parameters:

- channel** A numeric expression representing the sound channel to be used. The only valid parameters are 1, 2, or 3.
- volume** A numeric expression between zero and 15 representing the volume at which the sound is to be played. Zero is off, 15 is the highest volume.
- note** This is a numeric expression between 1 and 12 representing the note to be played.

<u>Number</u>	<u>Note</u>
1	C
2	C#
3	D
4	D#
5	E
6	F
7	F#
8	G
9	G#
10	A
11	A#
12	B

octave This is a numeric expression between 1 and 8 which represents the value of the octave to be played. One represents the lowest octave, 8 is the highest.

#period This optional parameter may be used in place of note and octave. The period is calculated as:

$$\text{period}=\text{TRUNC}(125000/\text{frequency}+0.5)$$

duration This is an integer expression and specifies the time in 1/50's of a second to wait before executing the next command in sequence.

Example:

```
REPEAT
  READ note%,oct%,dur%
  SOUND 1,15,note%,oct%,dur%*5
  SOUND 1,0
UNTIL n%=0
'
DATA 1,5,3
DATA 1,5,3
DATA 1,6,3
DATA 1,6,3
DATA 3,6,3
DATA 3,6,3
DATA 1,6,6
DATA 11,5,3
DATA 11,5,3
DATA 10,5,3
DATA 10,5,3
DATA 8,5,3
DATA 8,5,3
DATA 6,5,6
DATA 0,0,0
```

SPACE\$
Function

Syntax: **SPACE\$(*num_expr*)**

This function creates a character string containing a specified number of character spaces, ASCII 32.

Parameters:

num_expr Any integer expression between 0 and 32,767.

Example:

```
a$=SPACE$(32)
PRINT LEN(a$)
PRINT a$
```

SPC
Function

Syntax: **SPC(*num_expr*)**

This function can only be used in conjunction with the PRINT or LPRINT statements, and produces a specified number of spaces (ASCII 32) in the line to be printed.

Parameters:

num_expr Any integer expression between 0 and 256.

Example:

```
PRINT "Hello,";SPC(10)"there"
```

Related Words:

PRINT, LPRINT

SPOKE
Function

Syntax: **SPOKE *addr,num_expr***

Abbreviation: **SP**

Stores 1 byte in a designated area of main memory. This function operates in the Supervisor mode of the 68000 CPU, so reserved memory may be addressed. Otherwise, this command is identical to POKE.

Parameters:

addr The address in main memory for storing the number. The address must be an even number.

num_expr A number between 0 and 255 to be placed in memory.

Example:

```
a$="A"  
a%=VARPTR(a$)  
PEEK a%  
SPOKE a%,66  
PEEK a%  
PRINT a$
```

Related Words:

POKE, DPOKE, LPOKE, SDPOKE SLOPE

SPRITE

Function

Syntax:

SPRITE string_var[x_loc,y_loc]

Abbreviation:

SPR

Places a predefined 16 x 16 pixel sprite at a specified location on the screen, or deletes a sprite being displayed on the screen. For more detailed information, see Chapter 4, Computer Animation.

Parameters:

string_var Any string variable name. The string consists of 37 characters.

Character

- | | |
|------|---------------------------------|
| 1 | X-coordinate of the "hot spot" |
| 2 | Y-coordinate of the "hot spot" |
| 3 | 0 normal display mode |
| 1 | XOR against screen |
| 4 | Screen Color (usually 0) |
| 5 | Sprite Color |
| 6-37 | Bitpattern of screen and sprite |

The "hot spot" is a defined location within the bitpattern of the sprite that will be at the x,y position specified in the SPRITE function.

x_loc X coordinate of sprite's "hot spot"

y_loc Y coordinates of sprite's "hot spot"

The string variable defining a sprite is constructed by concatenating the numeric variables using MKI\$.

Example:

```
' Sprite-Convert data in string
LET gunsight$=MKI$(7)+MKI$(7)
LET gunsight$=gunsight$+MKI$(0)
LET gunsight$=gunsight$+MKI$(0)
LET gunsight$=gunsight$+MKI$(15)
FOR ctr%=1 TO 16
    READ foregrnd,backgrnd
    LET gunsight$=gunsight$+MKI$(backgrnd)+MKI$(foregrnd)
NEXT ctr%
HIDEM
CLS
DO
    x_loc%=MOUSEX
    y_loc%=MOUSEY
    EXIT IF MOUSEK
    VSYNC
    SPRITE gunsight$,x_loc%,y_loc%
LOOP
DATA 33026,0,16644,0,8456,0
DATA 4368,0,256,0,0,0,0,0,63550,0
DATA 0,0,0,0,256,0,4368,0,8456
DATA 0,16644,0,33026,0,0,0
```

SPUT**Function****Syntax:****SPUT** *string_variable***Abbreviation:****SPU**

Copies a 32,000 byte string, previously defined by using SGET, into screen memory.

Parameters:*string_variable* Any legal string variable name.**Example:**

```
PRINT AT(10,10);"Screen 1"  
SGET screen_1$  
CLS  
PRINT at(10,10);"Screen 2"  
SGET screen_2$  
CLS  
DO  
    SPUT screen_1$  
    PAUSE 50  
    SPUT screen_2$  
    PAUSE 50  
LOOP
```

Related Words:

GET, PUT, SGET

SQR**Function****Syntax:****SQR**(*num_expr*)

This function calculates the square root of any positive number.

Parameters:*num_expr* Any positive numeric expression.

Example:

```
INPUT "Enter a positive number";n%
PRINT "The Square root of ",n%," is ";SQR(n%)
```

STOP
Statement**Syntax:** **STOP****Abbreviation:** **ST**

Halts the execution of a program. STOP does not close any open files. Typing CONT will cause program execution to continue with the line immediately following the STOP statement. This is a handy statement for debugging.

Example:

```
FOR ctr%=1 TO 10
  PRINT ctr%
  STOP
NEXT ctr%
```

Related Words:
CONT, END, EDIT

STR\$
Function**Syntax:** **STR\$(*num_expr*)**

Converts a number or numeric expression into a string variable in decimal form.

Parameters:

num_expr Any number or numeric expression in any form. (Decimal, hexadecimal, octal, or binary form.)

Example:

```
a%=12345  
b=PI  
a$=STR(a%)  
b$=STR(b)  
c$=STR("123456")  
PRINT a%  
PRINT b%  
PRINT c%
```

Related Words:
HEX\$, OCT\$, BIN\$

STRING\$
Function

Syntax: **STRING\$(number,char_string)**
 STRING\$(number,code)

Creates a string by repeating an ASCII character or character string a specified number of times.

Parameters:

- number** A numeric expression between 0 and 32767. This number determines how many times the character in the string will be repeated.
- char_string** Any character string expression. The length of the string may not exceed 32,767 characters.
- code** ASCII code of the character to be repeated. An alternate method of expressing the character to be repeated.

Example:

```
c$="A"  
PRINT STRING$(75,c$)  
PRINT STRING$(40,"A")  
PRINT STRING$(50,66)  
PRINT STRING$(20,"AB")
```

SUB
Function**Syntax:** **SUB** *variable,expression***Abbreviation:** S

Subtracts a numeric expression from a numeric variable. This function is a fast way to provide subtraction.

Parameters:*variable* Any numeric variable.*expression* A numeric expression.**Example:**

```
DIM a%(10000)
FOR ctr%=1 TO 10000
  A%(ctr%)=10
NEXT ctr%
t=TIMER
PRINT (TIMER-t)/200
FOR ctr%= 1 TO 10000
  SUB a%(ctr%),5
NEXT ctr%
FOR ctr%=1 TO 10000
  A%(ctr%)=10
NEXT ctr%
t=TIMER
FOR ctr%=1 TO 10000
  a%(ctr%)=a%(ctr%)5
NEXT ctr%
PRINT (TIMER-t)/200
```

Related Words:

ADD, DIV, MUL

SWAP
Statement**Syntax:** **SWAP var_1,var_2****Abbreviation:** **SW**

Exchanges the values held in any two variables or array elements. The variables or array elements must be the same variable type.

Parameters:

var_1 Any numeric, numeric array, string, string array, boolean, or boolean array variable.

var_2 Any numeric, numeric array, string, string array, boolean, or boolean array variable.

Example:

```
a%=1  
b%=2  
PRINT "Variable a% = ";a%  
PRINT "Variable b% = ";b%  
PRINT "After SWAP,"  
SWAP a%,b%  
PRINT "Variable a% = ";a%  
Print "Variable b% = ";b%
```

SYSTEM
Statement**Syntax:** **SYSTEM****Abbreviation:** **SY**

Exits from the interpreter. Normally, control is returned to the GEM desktop.

Example:

```
PRINT "Enter password:";  
INPUT pass$  
IF pass$<>"OK"  
    SYSTEM  
ENDIF
```

Related Words:

END, QUIT

TAB**Statement****Syntax:****TAB(*int*)**

This statement is used in conjunction with the PRINT statement, and is similar to the TAB function of a typewriter; it moves the cursor a specified number of spaces to the right before printing. If the cursor is already past the tab position, the cursor is set to the specified position in the following line.

Parameters:*int* Any integer expression.**Example:**

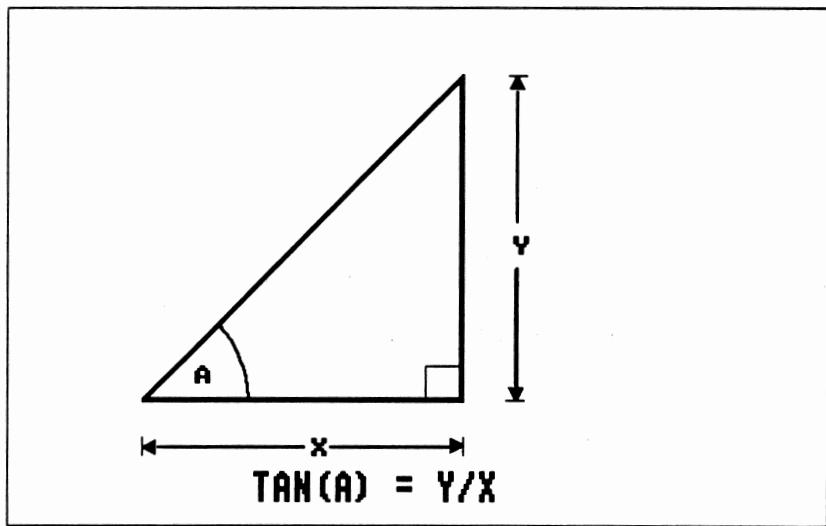
PRINT TAB(10); "Hello"

Related Words:

SPACE

TAN**Function****Syntax:****TAN(*angle*)**

Computes the trigonometric Tangent of an angle expressed in radians. The Tangent is defined as the ratio created by dividing the length of the side opposite the angle by the side adjacent to it.

Figure 2-9 Tangent Function**Parameters:**

angle Any numeric expression representing the angle. This value is expressed in radians. If the known angle is expressed in degrees, it must be expressed as:

$$\tan = \text{angle} * \text{PI}/180$$

Example:

```
INPUT "Angle (in radians)";angle  
PRINT TAN(angle)  
INPUT "Angle in degrees";angle  
PRINT TAN(angle*PI/180)
```

Related Words:

COS, SIN

TEXT

Statement

Syntax: **TEXT** *x_loc,y_loc,[int],string*

Abbreviation: T

Displays text on the screen at a specified location in graphics mode.

Parameters:

x_loc X location of the bottom left corner of the first character to be displayed.

y_loc Y location of the bottom left corner of the first character to be displayed.

int This optional parameter may be either positive or negative and specifies the length of the text to be displayed.

If the value is positive, the text is adjusted to the specified length by adjusting the spacing between the characters.

If the value is negative, the text is adjusted to the specified length by adjusting the space between words.

If the value is zero, or omitted, the text is displayed with its original spacing.

string Any string or string variable.

Example:

```
DEFTEXT 1,16,03  
TEXT 10,50,"Hello"
```

Related Words:

DEFTEXT

TIME\$
Function**Syntax:** *string* = TIME\$

Returns the system time in a string variable in the HH:MM:SS format.

Parameters:*string* Any string variable**Example:**t\$=TIME\$
PRINT t\$

or

PRINT TIME\$

Related Words:
DATE\$

TIMER
System Variable**Syntax:** *int_expr* = TIMER

This system variable contains the time elapsed since turning on the computer in 200ths of a second.

Parameters:*int_expr* Any integer expression.**Example:**t=TIMER
DO
 EXIT IF INKEY\$<>""
 z=INT(TIMER-t)/100
 PRINT AT(1,1)z
LOOP

This example simulates a stop watch, accurate to one hundredth of a second.

TITLEW**Function****Syntax:** **TITLEW** *int, string***Abbreviation:** TIT

Places a title in the title bar of a specified window. This function may be used before or after the window is opened with the OPENW function.

Parameters:

int A numeric expression representing the number of the window to which the title string is to be assigned.

Example:

```
TITLEW 1,"Get ready."
OPENW 1
PAUSE 100
TITLEW 1,"Get set."
PAUSE 100
TITLEW 1,"Go!"
PAUSE 50
DO
    TITLEW 1,TIME$
LOOP
```

Related Words:
OPENW, CLOSEW

TROFF**Command****Syntax:** **TROFF****Abbreviation:** TROF

Switches the TRACE command off.

Example:

```
TRON
FOR ctr%=0 TO 99
    PRINT ctr%
NEXT ctr%
TROFF
FOR ctr%=0 TO 99
    PRINT ctr%
NEXT ctr%
```

Related Words:

TRON

TRON

Command

Syntax: TRON[*num*]

Abbreviation: TR

Switches the trace command on. After this command is executed, all statements are shown prior to executing. Output maybe redirected to a disk file, or the printer. Output maybe slowed down by pressing the <CapsLock> key, or halted momentarily by pressing the right <Shift> key.

Parameters:

num A data channel.

Example:

```
OPEN "",#99,"LST:" !send output to printer
TRON #99
FOR ctr%=0 TO 99
    PRINT ctr%
NEXT ctr%
TROFF
CLOSE #99
FOR ctr%=0 TO 99
    PRINT ctr%
NEXT ctr%
```

Related Words:

TROFF

TRUE**System Variable****Syntax:** *boolean_exp = TRUE*

A way of expressing the value -1. Used to assign the value of TRUE to a boolean variable.

Parameters:*boolean_exp* Any boolean variable.**Example:**

```
PRINT "Press any key."
a=INP(2)
IF a>97 AND a<122
  b!=TRUE
ENDIF
IF b!
  PRINT "You pressed a letter key."
ELSE
  PRINT "You did not press a letter key."
ENDIF
```

Related Words:

FALSE

TRUNC**Function****Syntax:** **TRUNC(*num_expr*)**

Returns the integer portion of a numeric expression. This function is identical to INT for positive numbers. TRUNC is the complimentary function to FRAC.

Parameters:*num_expr* Any numeric expression.

Example:

```
a=1234.5678
PRINT TRUNC(a)
PRINT TRUNC(PI)
PRINT TRUNC(1.99)
PRINT TRUNC(13)
```

Related Words:
FIX, INT

TYPE
Function**Syntax:** **TYPE(*ptr*)**

This function determines the type of variable at which the pointer is set. A value is returned according to the type of variable the pointer is pointing at.

<u>Value</u>	<u>Type</u>
0	Real numeric variable
1	String variable
2	Integer variable
3	Boolean variable
4	Real array
5	String array
6	Integer array
7	Boolean array
-1	Error

Parameters:

ptr An integer expression.

Example:

```
DIM a$(5),a%(5),a(5),a!(5)
PRINT b$="Test"
PRINT b=12345
PRINT b%=12
PRINT b!=TRUE
PRINT TYPE(*a$)
PRINT TYPE(*a%)
PRINT TYPE(*a)
PRINT TYPE(*a!)
PRINT TYPE(*b$)
PRINT TYPE(*b)
PRINT TYPE(*b%)
PRINT TYPE(*b!)
```

UPPER\$
Function

Syntax: **UPPER\$(*string*)**

Converts all lower case characters in a string into upper case characters. All non-alphabetic characters are unchanged.

Parameters:

string Any character string expression.

Example:

```
a$="test string"
PRINT a$
b$=UPPER$(a$)
PRINT b$
```

Related Words:

LOWER\$

VAL**Function****Syntax:****VAL(*string_var*)**

Converts numbers written as strings into their numeric value in any form (decimal, hexadecimal, octal or binary). In the case of an alphabetic string or empty string, a value of zero is returned.

Parameters:

string_var An string expression or variable.

Example:

```
a$="12345"  
a=VAL(a$)  
PRINT a
```

Related Words:**VAL?****VAL?****Function****Syntax:****VAL?(*string_var*)**

Determines the number of characters in a string which can be converted into a numeric value by the VAL function. In the case of an alphabetic string, or an empty string, a value of zero is returned.

Parameters:

string_var Any string expression or string variable.

Example:

```
a$="12345"  
b$="$1234.56"  
PRINT VAL?(a$)  
PRINT VAL?(b$)  
PRINT VAL?("1234")
```

Related Words:**VAL**

VARPTR
Function

Syntax: **VARPTR(*variable*)**

Determines the address of the first byte of a variable.

Parameters:

variable Any variable type (numeric, string, or boolean variable.)

Example:

```
a$="test"  
a=12345  
a!=TRUE  
PRINT VARPTR(a$)  
PRINT VARPTR(a)  
PRINT VARPTR(a!)
```

Related Words:

ARRPTR

VDIBASE
System Variable

Syntax: *var* = VDIBASE

Determines the address for GEM storage above the area used by GFA BASIC. GEM stores parameters used for VDI functions in this area. PEEKs and POKEs can lead to interesting effects, and disastrous crashes. Experiment at your own risk. System crashes are common when tampering with this area.

VDISYS
Function

Syntax: **VDISYS** *number*
VDISYS(*number*)
VDISYS

This function permits accessing VDI (Virtual Device Interface) functions from within GFA BASIC. All parameters needed for the VDI call are entered in the VDI parameter block.

If the function number has not been entered in the CTRL block, the form of **VDISYS** *number* or **VDISYS**(*number*) may be used.

When the function number has been entered in the CTRL block, it is only necessary to use the form of **VDISYS**.

Parameters:

number VDI function number.

Related Words:
GEMSYS

VOID
System Variable

Syntax: **VOID** *expression*

Abbreviation: VO

A dummy parameter used for calling a function which performs an operation and discards the result. This is faster, especially when compiled, than assigning the value to a variable.

Parameters:

expression Any statement which returns a value.

Example:

VOID FRE(0) !Forced garbage collection

or

VOID INP(2) !Wait for a key

VSYNC
Function**Syntax:** **VSYNC****Abbreviation:** **VS**

Synchronizes the screen by causing the program to wait for next vertical blank before continuing. Same as calling XBIOS(37), the XBIOS VSYNC function. Useful for avoiding screen flickering during animation sequences using PUT and GET.

WAVE
Function**Syntax:** **WAVE voice,env,form,len,dur****Abbreviation:** **WA**

This function is used to provide access to multivoice music and sound capabilities of GFA BASIC. WAVE does not produce any sound, but defines the sound which will be produced by using the SOUND function.

Parameters:

voice An integer expression between 0 and 32 which determines which combination of the three sound channels will be turned on, by setting specific bits. 256 times the period of the noise generator (0 to 31) may also be added to the value of the voice.

<u>Decimal</u>	<u>Binary</u>	<u>Channel</u>
0	00000000	All channels OFF
1	00000001	Channel 1 ON
2	00000010	Channel 2 ON
4	00000100	Channel 3 ON
8	00001000	Channel 1 NOISE ON
16	00010000	Channel 2 NOISE ON
32	00100000	Channel 3 NOISE ON

env Any integer expression which defines the channels for which the envelope is active. This is also a bitwise value.

<u>Decimal</u>	<u>Binary</u>	<u>Channel</u>
1	00000001	Channel 1 Envelope ON
2	00000010	Channel 2 Envelope ON
4	00000100	Channel 3 Envelope ON

form An integer expression which specifies the shape of the wave.

<u>Value</u>	<u>Description</u>
0-3	Linear, falling
4-7	Linear, rising, terminating
8	Sawtooth, rising
9	Linear, falling
10	Peaked, begin falling
11	Linear, falling, variable volume
12	Sawtooth, rising
13	Linear, rising, continuous
14	Peaked, begin rising
15	Linear rising

len An integer expression which specifies the length of time during which the envelope is to be played.

dur An integer expression which specifies the time, in 1/50'ths of a second, to delay before executing the next command

Any parameter which is to remain unchanged can be omitted. The statement WAVE 0,0 may be used to turn off all sound channels.

Example:

```
SOUND 1,15,1,4,20
SOUND 2,15,4,4,20
SOUND 3,15,8,4,20
WAVE 7,7,0,65535,300
WAVE 0,0
```

WHILE...WEND**Statement****Syntax:**

WHILE *condition*
 program statements
WEND

Abbreviation:

W
WE

This function creates a loop which is executed repeatedly while a specified condition is true. The WHILE loop must be closed with a WEND statement. The condition is checked before the loop is executed, so it is possible to define a WHILE...WEND loop which is never executed.

Parameters:

condition Any numeric or logical expression or variable. If this value is evaluated as true, the program loops between the WHILE and WEND statements.

program statements Any block of valid program statements.

Example:

```
WHILE x%<10
  INC x%
  PRINT x%
WEND
```

Related Words:**REPEAT...UNTIL**

WINDTAB**System Variable****Syntax:**

WINDTAB

This system variable contains the address of the window parameter table. The parameters for window control are stored in 16-bit words.

These parameters may be defined or changed by DPOKEing values to the appropriate offset of WINDTAB, as illustrated in Table 2-2.

Table 2-2 Window Attributes

<u>Offset</u>	<u>Description</u>
0	Identification number (handle) of window.
2	Window attributes. Properties of a window, such as a close box, full box, move bar, etc., may be defined by DPOKEing the sum of the desired values into this address.
<u>Value</u>	<u>Property</u>
0	No attributes
1	Title line
2	Close box
4	Full box
8	Move bar
16	Information line
32	Size gadget
64	Up arrow
128	Down arrow
256	Vertical slider
512	Left arrow
1024	Right arrow
2048	Horizontal slider
4	Window X coordinate
6	Window Y coordinate
8	Window width
10	Window height
12-22	Same as 0-10 for window 2
24-34	Same as 0-10 for window 3
36-46	Same as 0-10 for window 4
38	Dummy parameter (1)
40	Dummy parameter (0)
42	Screen X coordinate
44	Screen Y coordinate
46	Screen width
48	Screen height
50	X intersection of four windows
52	Y intersection of four windows
54	X origin of graphics commands
56	Y origin of graphics commands

WRITE
Statement**Syntax:** **WRITE** *expr_list***Abbreviation:** WR

This statement displays data on the screen in a special format. The only valid punctuation is the semicolon (;), after which additional data will be displayed without any separating spaces. The data is displayed separated by commas; strings are enclosed in quotes.

Parameters:*expr_list* A list of valid variables or expressions to be displayed.**Example:**

```
a$="Hello"  
b=1  
c=2  
d=3  
WRITE a$  
WRITE b,c,d
```

Related Words:
PRINT, INPUT, WRITE#

WRITE#
Statement**Syntax:** **WRITE#** *file, expr_list***Abbreviation:** WR #

This statement sends data to a disk file in a special format, suitable for retrieving with INPUT#. The data is separated by commas and strings are enclosed in quotes.

Parameters:*file* The channel number for the file as assigned by the OPEN statement.*expr_list* A list of valid variables or expressions to be displayed.

Example:

```
OPEN "O",#1,"TEST.DAT"
a$="Hello"
b=1
c=2
d=3
WRITE#1,a$
WRITE#1,b,c,d
CLOSE#1
OPEN "I",#1,"TEST.DAT"
INPUT#1,a$,b,c,d
CLOSE#1
PRINT a$
PRINT b
PRINT c
PRINT d
```

Related Words:

INPUT, WRITE, PRINT, PRINT#

XBIOS

Function

Syntax:

XBIOS(*function*[*parameter_list*])

This function permits access to the XBIOS functions of the operating system.

Parameters:

function An integer expression representing the number assigned to the XBIOS function being called. Refer to Appendix C, XBIOS Functions for more information.

parameter_list List of parameters required by the XBIOS function being called.

Example:

rez%=XBIOS(4) !Get screen resolution

Related Words:

BIOS, GEMDOS

XOR

Conditional Operator

Syntax: *condition XOR condition*

Provides an exclusive OR of the specified conditions. Does not evaluate conditions, but logically compares the outcome of two conditions. XOR requires the outcome of the two conditions to be different.

Condition	XOR	Condition	Result
-1 (TRUE)	XOR	-1 (TRUE)	0 (FALSE)
-1 (TRUE)	XOR	0 (FALSE)	-1 (TRUE)
0 (FALSE)	XOR	-1 (TRUE)	-1 (TRUE)
0 (FALSE)	XOR	0 (FALSE)	0 (FALSE)

Parameters:

condition Any logical expression, arithmetic equation, or comparison.

Example:

```
a$="A"
f$="F"
IF a$="A" XOR f$="B"
    PRINT "XOR satisfied"
ELSE
    PRINT "XOR not satisfied"
ENDIF
```

Related Words:

AND, OR, NOT, IMP, EQV

The following list illustrates the GFA BASIC command syntax with required parameters. Command line parameters placed between brackets [] are optional and not required for the command.

Variable Names
A variable must begin with a letter, but may consist of any combination of letters and numbers, up to a maximum length of 255 characters.

Types of Variables
var Real variable, a six byte floating point value, accurate to 11 decimal places. In Scientific Notation, exponents of up to 154 are permitted.

var% Integer variable, a four byte value representing a whole number between -2147483648 and +2147483647

var! Boolean variable. Requires 2 bytes for storage and may only represent TRUE (-1) or FALSE (0) values.

var\$ String variable. May contain up to 32,767 alpha-numeric characters.

Array Variables
var(i) One dimension Real array.
var%(i) One dimension Integer array.
var!(i) One dimension Boolean array.
var\$(i) One dimension String array.

Pointers
***var** Real pointer
***var%** Integer Pointer
***var!** Boolean Pointer
***var\$** String Pointer
***var()** Array Pointer

Constants
ADDRIN Addr of AES address input block. (Pg. 37)
ADDROUT Addr of AES address output block. (Pg. 37)
BASEPAGE Addr of the basepage of GFA BASIC. (Pg. 43)
CONTRL Addr of VDI control block. (Pg. 67)
FALSE 0 (Pg. 103)
GB Addr of AES control block.
GCONTROL Addr of AES control block.
GINTIN Addr of AES integer input block.
GINTOUT Addr of AES integer output block.
HIMEM Addr of area in memory not required by GFA BASIC. (Pg. 122)
INTIN Addr of VDI integer input block. (Pg. 133)
INTOUT Addr of VDI integer output block. (Pg. 133)
PI 3.1415926536 (Pg. 187)
PTSin Addr of VDI point input block. (Pg. 201)

PTSOUT Addr of VDI point output block. (Pg. 202)
TRUE -1 (Pg. 241)
VDIBASE Addr of GEM storage area used by GFA BASIC. (Pg. 245)
WINDTAB Addr of window parameter table. (Pg. 249)

Operators
AND Logical conjunction, bit-wise operation. (Pg. 38)
DIV Integer division. (Pg. 90)
EQV Logical equivalence. (Pg. 96)
IMP Logical implication. (Pg. 123)
MOD Returns remainder from Integer division (Pg. 158)
NOT Logical negation, bit wise operation. (Pg. 164)
OR Logical comparison, bit wise operation. (Pg. 179)
XOR Logical exclusive OR, bit wise operation. (Pg. 253)

Mathematical Functions
ABS(x) Returns the absolute value. (Pg. 36)
ADD x,y Increase the value of *x* by *y*. (Addition) (Pg. 36)
ATN(r) Calculates Arc-tangent of an angle in radians. (Pg. 41)
COS(n) Calculates the Cosine in radians of a number. (Pg. 67)
DEC x,y Decrements *x* by *y*. (Pg. 74)
DIV x,y Divides *x* by *y*. (Pg. 90)
EVEN(x) Returns a -1 value for an even number; otherwise returns 0. (Pg. 100)
EXP(n) Calculates value of an exponent. (Pg. 103)
FIX(x) Returns the integer portion of a value. (Pg. 107)
FRAC(x) Returns the fractional portion of a value. (Pg. 111)
INC x,y Increments *x* by *y*. (Pg. 124)
INT(x) Returns the integer portion of a value. (Pg. 132)
LOG(x) Returns the natural logarithm of a value. (Pg. 141)
LOG10(x) Returns the base 10 logarithm of a value. (Pg. 142)
MUL x,y Multiplies *x* by *y*. (Pg. 162)
ODD(x) Returns a -1 value for an odd number; otherwise returns 0. (Pg. 165)
RANDOM(x) Returns a random integer number between 0 and *x*. (Pg. 204)
RND(x) Returns a random value between 0 and 1. (Pg. 215)

SIGN(x)	Determines the sign of a value. (Pg. 222)	TRON	Enables trace mode. (Pg. 240)
<hr/>			
FOR var= [DOWNTO] [STEP] [i]	Looping Structures Statement	FOR	var= [DOWNTO] [STEP] [i] Looping Structures Statement
SQR(n)	Calculates the square root. (Pg. 230)	SQRT	x/y Calculates the square root of x by y. (Pg. 108)
SIN(angle)	Calculates the Sine in radians of a number. (Pg. 224) 0 = Zero -1 = Negative (x<0)	SIN	(angle) Calculates the Tangent of an angle in radians. (Pg. 233) Decreases the value of x by y. (Pg. 118)
TRUNC(x)	Returns the integer portion of a number. (Pg. 241) Returns the integer portion of a value without returning a value. (Pg. 246)	TRUNC	x/y Defines listing format. In this case, Commands, Functions, and Variables names are written with the first letter capitalized. (Pg. 80) Defines listing format. In this case, Commands and Functions are written in all capital letters. (Pg. 93)
CLEAR	Clears all variables and arrays. (Pg. 62)	CLEAR	END Terminates program and returns to the Editor. (Pg. 93) Deletes all files and terminates program. (Pg. 95)
CONT	Resumes program execution. (Pg. 65)	CONT	END Terminates program and returns to the Editor. (Pg. 93) Deletes all files and terminates program. (Pg. 95)
DEFLIST 1	Defines listing format. In this case, Commands, Functions, and Variables names are written with the first letter capitalized. (Pg. 80) Defines listing format. In this case, Commands and Functions are written in all capital letters. (Pg. 93)	DEFLIST 1	DEFLIST 0 Defines listing format. In this case, Commands, Functions, and Variables names are written with the first letter capitalized. (Pg. 80) Defines listing format. In this case, Commands and Functions are written in all capital letters. (Pg. 93)
EDIT	Edits an array and releases memory on a disk in ASCII format. (Pg. 137)	EDIT	ERASE(filed) Deletes an array and releases memory on a disk in ASCII format. (Pg. 137)
LIST	Lists program presently in memory on a disk. (Pg. 137)	LIST "CON;"	LIST "FILE;" Sends listing of program currently in memory on the output screen. (Pg. 137)
NEW	Removes program from memory. (Pg. 138)	NEW	LIST "FILE;" Sends listing of program presently in memory on a disk. (Pg. 137)
QUIT	Removes binary value. (Pg. 163)	QUIT	LIST "FILE;" Sends listing of program presently in memory on a disk. (Pg. 137)
RUN	Precedes Detach. (Pg. 204)	RUN	REM, ! Enables remarks to be included in GEM desktop. (Pg. 209)
STOP	To GEM desktop. (Pg. 234)	STOP	Detaches trace mode. (Pg. 239)

CVD(a\$)	Converts an 8-byte character string in MBASIC format into a number. (Pg. 69)	RIGHT\$(a\$,n)	Returns all characters beginning with the <i>n</i> th character from the right side of a string. (Pg. 214)
CVF(a\$)	Converts a 6-byte character string in GFA BASIC format into a number. (Pg. 70)	RSET var = string	Right justifies a string. (Pg. 216)
CVI(a\$)	Converts 2-byte character string into 16 bit integer value. (Pg. 70)	SPACES(n)	Creates a string of <i>n</i> spaces. (Pg. 226)
CVL(a\$)	Converts a 4-byte character string in a 32-bit integer value. (Pg. 71)	STR\$(a)	Transforms a decimal value into a character string. (Pg. 231)
CVS(a\$)	Converts a 4-byte Atari BASIC character string into a 32-bit integer value. (Pg. 72)	STRING\$(n,"c")	Creates a string of <i>n</i> characters specified by the included string. (Pg. 232)
FRE(0)	Returns free storage space in bytes. (Pg. 111)	STRING\$(n,c)	Creates a string of <i>n</i> characters specified by the ASCII code <i>c</i> . (Pg. 232)
HEX\$(a)	Transforms a value into a character string expression in hexadecimal (base 16) form. (Pg. 121)	UPPERS(a\$)	Converts a string to all upper case characters. (Pg. 243)
OCT\$(a)	Transforms a value into a character string expression in octadecimal (base 8) form. (Pg. 165)	VAL(a\$)	Returns the numerical value of the first character in a string. (Pg. 244)
INSTR([a\$,b\$,n])	Returns the position of a specified character within a string. (Pg. 132)	VAL?(a\$)	Returns the number of characters in a string which can be converted into a numerical value. (Pg. 244)
INSTR([n,]a\$,b\$)	Returns the position of a specified character within a string. (Pg. 132)		
LEFT\$(a\$,n)	Returns all characters beginning with the <i>n</i> th character from the left side of a string. (Pg. 134)		
LEN(a\$)	Returns the length of a string. (Pg. 134)		
LSET var = string MID\$(a\$,n1[,n2])	Left justifies a string. (Pg. 145) Returns the character(s) from the position(s) specified by <i>n1</i> and <i>n2</i> from within a string. (Pg. 152)		
MID\$(a\$,n1[,n2])=b\$	Inserts a specified character or group of characters into a string, at the positions specified by <i>n1</i> and <i>n2</i> . (Pg. 153)		
MKD\$(i)	Converts a number into an MBASIC compatible 8-byte format. (Pg. 155)		
MKF\$(i)	Converts a number into GFA BASIC 6 byte format. (Pg. 156)		
MKI\$(i)	Converts a 16-bit integer into a 2-byte string. (Pg. 156)		
MKL\$(i)	Converts a 32-bit integer into a 4-byte string. (Pg. 157)		
MKS\$(i)	Converts a number into an Atari BASIC compatible 4-byte format. (Pg. 157)		
			Fields and Pointers
		ARRAYFILL field0,n	Assigns the value, <i>n</i> , to all elements in an array. (Pg. 39)
		ARRPTR(var)	Returns the address of the descriptor of a string or an array. (Pg. 40)
		DIM(a\$,x)	Sets dimensions of an array or arrays. (Pg. 86)
		DIM?(field)	Returns the number of elements in an array. (Pg. 87)
		MAX(expr)	Returns the greatest value from a list of expressions. (Pg. 146)
		MIN(expr)	Returns the least value from a list of expressions. (Pg. 154)
		OPTION "Text"	Passes control commands to the GFA Compiler. (Pg. 177)
		OPTION BASE 0	Sets the lower limit of an array to zero. (Pg. 178)
		OPTION BASE 1	Sets the lower limit of an array to one. (Pg. 178)
		SWAP *ptr,foar()	Exchanges the elements in the field pointed to by <i>*ptr</i> with the elements in the array <i>foar()</i> . (Pg. 234)
		SWAP foar1(),foar2()	Exchanges the values of the elements in the array <i>foar1()</i> with the elements of <i>foar2()</i> . (Pg. 234)
		SWAP var1,var2	Exchanges values of <i>var1</i> with <i>var2</i> . (Pg. 234)

TYPE(*var)	Returns the type of variable at which the pointer is set. (Pg. 242)	INP?(n)	3 MID: (Midi port) Determines input status of specified peripheral device.(As shown above.) (Pg. 128)
	-1 = Error 0 = Real 1 = String 2 = Integer 3 = Boolean 4 = Real array 5 = String array 6 = Integer array 7 = Boolean array	INPUT	Permits information to be entered from the keyboard during program execution. (Pg. 128)
VARPTR(var)	Returns the address or starting address of a variable. (Pg. 245)	INPUT#	Permits information to be entered from a specified channel during program execution. (Pg. 130)
		INPUT\$(len,n\$)	Reads <i>len</i> characters from the keyboard or from a file. (Pg. 131)
		LINE INPUT	Permits a string to be entered during program execution. (Pg. 136)
		LINE INPUT#	Permits a string to be entered from a specified channel during program execution. (Pg. 137)
		LPOS(n)	Returns the column where the printer head is located. (Pg. 143)
		OUT n,c	Transfers one byte to specified output device. (Pg. 180)
			0 LST: (printer) 1 AUX: (RS-232) 2 CON: (Keyboard) 3 MID: (Midi port) 4 IKB: (Caution) 5 VID: (Screen)
		OUT?(n)	Returns status of specified output device. (Pg. 181)
		POS(n)	Returns the column in which the cursor is located. <i>n</i> is a dummy argument. (Pg. 193)
		PRINT or LPRINT or PRINT#	Displays information on the screen or specified data channel. (Pg. 144, 195, 197)
		AT(x,y)	Prints at a position specified by the cursor column and row. (Pg. 196)
		TAB(n)	Moves cursor position to the <i>n</i> th column. (Pg. 235)
CRSCOL	Input and Output Determines the cursor column. (Pg. 68)		
CRSLIN	Determines the cursor row. (Pg. 68)		
DEFNUM n	Rounds all numbers to <i>n</i> digits before output. (Pg. 84)		
FORM INPUT	Permits the input of a character string during program execution. (Pg. 109)		
FORM INPUT n AS a\$	Permits a character string to be changed during program execution. <i>n</i> is the max length of the string variable. (Pg. 110)		
HARDCOPY	Sends screen contents to an attached printer. (Same as pressing the <Alternate><Help> key combination. (Pg. 120)		
INKEY\$	Reads a character from the keyboard. (Pg. 125)		
INP(n)	Reads one byte at a time from an input device. (Pg. 126)		
	0 LST: (printer) 1 AUX: (RS-232) 2 CON: (Keyboard)		

SPC(<i>n</i>)	Prints a number of spaces as specified by <i>n</i> . (Pg. 227)	SEEK#<i>n,x</i>	Sets the file pointer on the <i>x</i> th byte of the file associated with channel number <i>n</i> . (Pg. 219)
;	Carriage return/line feed suppressed after printing.	WRITE#<i>n</i>	Stores data in a sequential file. (Pg. 251)
,	Carriage return/line feed suppressed. Cursor positioned at intervals of 16 columns. (17,33,49...)		
,	One space is left in for each apostrophe.		
PRINT USING	Formatted output of digit and character strings. (Pg. 198)		
WRITE	Outputs data to the screen. (Pg. 251)		
WRITE#	Stores data in a sequential file. (Pg. 251)		
<hr/>			
Sequential and Random Data			
CLOSE#<i>n</i>	Closes a data channel. (Pg. 63)	BGET #<i>n,adr,len</i>	Reads from a specified data channel directly to memory. (Pg. 45)
EOF#(<i>n</i>)	Determines whether the file pointer has reached the end of the file. (Pg. 96)	BLOAD "file" <i>adr</i>	Loads a disk file directly to a specified area of memory. (Pg. 50)
FIELD #<i>n,expr,AS</i>	Divides a record into arrays. (Pg. 104)	BPUT #<i>n,adr,len</i>	Writes a specified area of memory directly to disk. (Pg. 53)
GET#<i>n</i>	Reads a record from a random access file. (Pg. 115)	BSAVE "file" <i>adr,len</i>	Saves a specified area of memory to disk. (Pg. 54)
INP#(<i>n</i>)	Reads one byte of data from a file on data channel <i>n</i> . (Pg. 126)	CHAIN "file.bas"	Loads a program file into memory and starts the program. (Pg. 57)
INPUT#<i>n</i>	Obtains data from file on channel <i>n</i> . (Pg. 130)	EXEC (<i>flag,file.cmd,env</i>)	Loads and executes machine language or compiled programs from disk. (Pg. 101)
INPUT\$(<i>l</i>)	Reads a specified number of characters from the keyboard. (Pg. 131)	EXIST("file.ext")	Determines whether a specified file is present on the disk. -1 if file is present. 0 if not present. (Pg. 101)
LINE INPUT#<i>n,AS</i>	Obtains data from channel <i>n</i> . (Pg. 137)	KILL "file"	Deletes a disk file. (Pg. 133)
LOC#<i>n</i>	Returns the location of the file pointer for the file with the channel number <i>n</i> . (Pg. 139)	LIST "file.lst"	Saves a file to disk in ASCII format. (Pg. 137)
LOF#(<i>n</i>)	Returns the length of the file with the channel number <i>n</i> . (Pg. 140)	LOAD "file"	Loads a file into memory. (Pg. 138)
OPEN "mode" #<i>n,"file name",len</i>	Opens a data channel to a disk. Mode Open for: A Append I Input O Output R Random access U Reading and Writing (Pg. 175)	NAME "oldfile"	AS "newfile" Renames a file as specified. (Pg. 163)
OUT#<i>n,c</i>	Transfers one byte of data to a file on data channel <i>n</i> . (Pg. 180)	PSAVE "file"	Saves a file in protected format. (Pg. 201)
PRINT#<i>n</i>	Sends data to a file on data channel <i>n</i> . (Pg. 197)	SAVE "file"	Saves a file to disk. (Pg. 217)
PUT#<i>n</i>	Writes a record to a random access file. (Pg. 203)		
RELSEEK#<i>n,x</i>	Moves the file pointer <i>x</i> bytes in a file with the channel number <i>n</i> . (Pg. 208)		
<hr/>			
Directory Functions			
CHDIR "name"	Changes directories. (Pg. 58)	CHDRIVE <i>n</i>	Sets the default disk drive. (Pg. 58)
DFREE(<i>n</i>)	Returns the amount of free storage space remaining on the specified disk. (Pg. 86)	DIR "A:*.*"	Lists files on a disk. (Pg. 88)
DIR\$(<i>n</i>)	Returns the name of the active directory for drive number <i>n</i> . (Pg. 89)	FILES "A:*.*"	Lists files on a disk. (Pg. 105)
		FILESELECT "filespec","filename",<i>x\$</i>	Calls the GEM file selector box. (Pg. 106)
		MKDIR "name"	Creates a new directory folder. (Pg. 155)
		RMDIR "name"	Removes a directory folder. (Pg. 215)

Graphics Commands	
BITBLT \$%((),p%())	Marks n corner points. (Pg. 192)
POLYMARK n,x(),y()	Marks n corner
MOVE src,dst	Places a string obtained with rounded corners. (Pg. 194)
PRBOX x1,y1,x2,y2	Draws a rectangle with rounded corners. (Pg. 194)
PUT x1,y1,z\$	Places a string obtained with memory. (Pg. 194)
BOX x1,y1,x2,y2	Fast movement of blocks of memory. (Pg. 194)
COLOR c	Clears the screen. (Pg. 65)
CLS	Defines color corners. (Pg. 2206)
SETCOLOR i,n	Defines color components. (Pg. 2206)
SPCTA\$	Defines red, green and blue color components. (Pg. 219)
SPCTA\$	Places a rectangle with a 32k byte string. (Pg. 227)
SPUT A\$	Places a rectangle with a 32k byte string to the screen at the specified by x and y. (Pg. 228)
TEXT x,y,A\$	Places text on the screen at the position specified by x and y. (Pg. 247)
VSYNC	Wait for vertical blank. (Pg. 247)
HIDEM	Hides mouse (Pg. 121)
MOUSE x,y,t	Returns the mouse position (x,y) and status of the mouse buttons (W, (Pg. 159))
MOUSEK	Returns status of mouse buttons. (Pg. 160)
MOUSE	Left button down (Pg. 160)
MOUSE	Right button down (Pg. 160)
MOUSE	None pressed (Pg. 160)
SHOWN	Shows the mouse. (Pg. 161)
MOUSEY	Returns vertical coordinate of the mouse pointer. (Pg. 160)
MOUSEX	Returns horizontal coordinate of the mouse pointer. (Pg. 160)
SHOWM	Show mouse. (Pg. 161)
LINE x1,y1,x2,y2	Connects two or more points with straight lines. (Pg. 93)
ELLIPSE x,y,rw,ry	Draws and connects two or more points with straight lines. (Pg. 93)
DRAW x1,y1 [TO] x2,y2	Reads a rectangle from the screen and stores it in a screen and transparent mode. (Pg. 115)
GRAPHMODE n	Sets transparent mode. (Pg. 115)
GET x1,y1,x2,y2,z\$	Reads a rectangle from the screen and stores it in a transparent mode. (Pg. 115)
FILL x,y	2 Transparency
FILL x,y,rw,ry	1 Replace
3 XOR	3 XOR
4 Reverse transparent	4 Reverse transparent
PIRCLE x,y,r/s	Draws a filled rectangle. (Pg. 182)
BOX x1,y1,x2,y2	Draws a filled circle or arc. (Pg. 183)
POINT(x,y)	Draws a single point on the screen. (Pg. 187)
POLYFILL n,x(),y()	Retrurns the color value of a specified pixel location. (Pg. 188)
POLYLINE n,x(),y()	Draws a filled-in shape with corner points. (Pg. 191)

TIME\$	Returns the system time as a character string in the format hh:mm:ss. (Pg. 238)	ON MENU KEY GOSUB name	Defines procedure which handles keyboard input. (Pg. 169)
TIMER	Returns the time since the system was turned on in 200ths of a second. (Pg. 238)	ON MENU MESSAGE GOSUB name	Defines procedure for handling of GEM messages. (Pg. 170)
<hr/>			
Windows and Menus			
ALERT icon, text string, button, button text, a	Creates an Alert box. (Pg. 37) If <i>icon</i> = 0 Displays Nothing. 1 Exclamation point (!). 2 Question mark (?). 3 Stop sign.	ON MENU OBOX a,x,y,b,h GOSUB name	Defines the procedure for when the mouse moves out of a defined rectangular area. (Pg. 171)
CLEARW n	Clears the contents of the window specified by <i>n</i> . (Pg. 62)	OPENW 0	Opens the whole screen as a window without a menu bar. (Pg. 176)
CLOSEW 0	Switches to normal screen display. (Pg. 64)	OPENW n,x,y	Opens a specified window at the coordinates <i>x,y</i> . (Pg. 176)
CLOSEW n	Closes the window specified by <i>n</i> . (Pg. 64)	TITLEW n,"title"	Assigns a new name to the window specified by <i>n</i> . (Pg. 239)
FULLW n	Enlarges window <i>n</i> to full screen size. (Pg. 112)	<hr/>	
INFOW n, "info"	Defines a new information line for the specified window by <i>n</i> . (Pg. 124)	Machine Level Commands	
MENU field()	Creates menu bar. (Pg. 146)	BIOS(n,parameter list)	Permits calling TOS BIOS functions. (Pg. 47)
MENU KILL	Deactivates menu bar. (Pg. 148)	C:var(parameters)	Calls a compiled C or machine language sub-program located in memory at the address specified by <i>var</i> . Parameters are transferred as in C. (Pg. 55)
MENU n,x	Alters menu items. (Pg. 146) 0 Removes checkmark. 1 Places checkmark. 2 Writes in plain letters, cannot be selected. 3 Writes in normal letters, can be selected.	CALL [(parameters)]	Calls a machine language program located in memory at the address specified by <i>var</i> . (Pg. 56)
MENU OFF	Displays menu titles in "normal" mode. (Pg. 148)	DPEEK addr	Returns the contents of two bytes of memory. (Pg. 91)
MENU(f)	Detects events for ON MENU GOSUB. (Pg. 149)	DPOKE addr,n	Writes two bytes to addresses in memory. (Pg. 92)
ON MENU	Handles menu selection. (Pg. 168)	GEMDOS(n,parameter list)	Permits calling TOS GEMDOS functions. (Pg. 113)
ON MENU BUTTON c,m,s GOSUB name	Defines procedure for handling mouse button presses. (Pg. 172)	GEMSYS	Calls the AES function specified by the function number entered in the GCTRL block. (Pg. 114)
ON MENU GOSUB name	Defines procedure which handles menu selection. (Pg. 168)	GEMSYS (n)	Calls AES function specified by <i>n</i> . (Pg. 114)
ON MENU IBOX a,x,y,b,h GOSUB name	Defines the procedure for when the mouse moves into a defined rectangular area. (Pg. 171)	LPEEK (addr)	Returns the contents of four bytes from an address in memory. (Pg. 142)
		LPOKE addr,n	Writes four bytes to an address in memory. (Pg. 143)
		MONITOR	Calls a memory resident monitor program or a commands extension. (Pg. 158)
		PEEK addr	Returns the contents of one byte of memory. (Pg. 184)
		POKE addr,n	Writes one byte to an address in memory. (Pg. 188)
		RESERVE n	Increase or decreases the amount of memory available to GFA BASIC. (Pg. 210)

SDPOKE <i>addr,n</i>	Writes two bytes to an address in memory in the 68000's Supervisor mode. (Pg. 218)
SLPOKE <i>addr,n</i>	Writes four bytes to an address in memory in the 68000's Supervisor mode. (Pg. 224)
SPOKE <i>addr,n</i>	Writes one byte to an address in memory in the 68000's Supervisor mode. (Pg. 227)
VDISYS	Calls VDI function with the function number entered in the CONTRL block. (Pg. 246)
VDISYS(n)	Calls VDI function without entering the function number in the CONTRL block. (Pg. 246)
XBIOS(n)	Calls an XBIOS function. (Pg. 252)

ON BREAK

ON BREAK Deactivates ON BREAK GOSUB, or restore break conditions. (Pg. 166)

ON BREAK CONT Deactivates <Control> <Alternate> <Shift> break key sequence. ON BREAK may be used to reactivate keys. (Pg. 166)

ON BREAK GOSUB *proc_name* Jumps to a procedure when a break key sequence is encountered. (Pg. 166)

ON expression GOSUB *proc_list* Permits program redirection to a list of procedures. (Pg. 166)

Error Handling

ERR Returns the error code of any error which may have occurred. (Pg. 98)

ERROR n Simulates the occurrence of an error with the specified error code (*n*). (Pg. 100)

FATAL Returns the value 0 or -1 depending on which type of an error was encountered. Generally, -1 is an unrecoverable error. (Pg. 104)

ON ERROR Switched to normal error handling methods if an alternate means was previously defined. (Pg. 167)

ON ERROR GOSUB *proc_name* Defines an error handling procedure. (Pg. 167)

RESUME Defines method for exiting an error handling routine. (Pg. 212)

Chapter 3

GFA Graphics



During the late 1950s and early 1960s, the Massachusetts Institute of Technology (MIT) was the center of the computer world, an academic community actively involved in the research and construction of supercomputer systems.

In 1963, Ivan Sutherland, a young graduate student working on his doctoral dissertation at MIT, could often be seen working far into the night at the terminal of a giant mainframe computer. He wasn't alone in his labors. Other students doing research on computers had to use the late night hours, when the mainframes weren't being used for other purposes, for exploring the limits of this fledgling science of computers; the technology for microcomputers had not even been developed.

Sutherland wrote and rewrote his lines of code. After much revision, he was finally successful. The monitor display cleared, and a green line appeared: The first computer graphic, a straight line, had been produced by the giant mainframe. Not much by today's standards, but this occasion marked the beginning of a whole new art form, computer graphics.

This single straight line evolved into the Sketchpad line-drawing program. Sutherland's early drawing program allowed the user to point at the screen with a light pen to sketch objects. This simple system was the predecessor of today's complex computer-aided drawing (CAD) programs.

A lot had to happen to computers, however, before sophisticated computer graphics could become a reality. For a while computers remained the expensive tools of government and universities. In 1965, IBM introduced the first mass-produced cathode ray tube (CRT), but the price was more than \$100,000 for the CRT only. This price tag made entry into computer graphics practically impossible for all but a few specialized users. In 1968, Tektronics introduced the storage-tube CRT, which displayed a drawing by retaining the image until the user replaced it. This type of display eliminated the need for costly memory and hardware to store and display the image, but the selling price of \$15,000 still placed the CRT far beyond the means of most users.

It wasn't until the late 1970s and early 1980s that the microcomputer industry exploded, and computer wars brought the price and power of the personal computer within the reach of the middle-income person.

Soon, powerful graphics programs such as DEGAS™, GFA Artist™, and NEochrome™ had introduced many people to this new art form, and some even discovered hidden talents. And now that computer art has become accessible to so many people, those people want to make the best possible use of its potential.

In this chapter, we'll take you beyond DEGAS™ and NEochrome™, showing you how to develop and use the powerful graphics routines built into the ST. Previously, exploitation of these features was rather difficult. Most programmers resorted to C or assembly language programming when sophisticated graphics were required. GFA BASIC delivers the power to fully utilize the graphics potential of your ST. Stunning graphics are easily obtained using the GFA BASIC interpreter, while the GFA BASIC compiler allows you to develop arcade-quality games in BASIC, then execute the programs from the GEM Desktop with your program running at speeds rivaling those of assembly language.

Computer graphics is a generic term which refers to any image created with a computer. This covers a wide range, from a simple line on a monitor, to intricate pictures developed with drawing programs, and more. Some computer scientists, for

instance, are using computers to control lasers which produce holographic images. Perhaps the scene from Star Wars where C3PO and Chew-baka are playing a chesslike game with holographic images isn't even that far-fetched.

The motion picture industry, in recent years, has used computer graphics extensively. The transformation of the Genesis planet in Star Trek II, The Wrath of Khan, was an effect produced with fractal graphics -one of the most fascinating areas of computer graphics (and one which we'll explore later in this chapter). Disney Studios merged real action with computer-generated graphics in the movie TRON. The Last Starfighter has 25 minutes of spectacular computer imagery, with each frame requiring dozens of algorithms and hundreds of intricate designs. In fact, such movies have contributed to the birth of a new industry devoted to enhancing computer-generated graphics technology, with one of the leaders in this new industry being Lucasfilm, a pioneer in special effects since the Star Wars trilogy.

Although holographic imaging and Hollywood-type special effects are beyond the scope of this book, it will show you a fascinating world open to those who know the power of the ST.

Graphics Capabilities

Good graphics require a minimum of a high-resolution display and a large number of colors to work with. Some rumors are beginning to circulate about computers for homes and businesses with monitors capable of displaying up to one million picture elements (or pixels). Currently only dedicated graphics workstations, such as the Sun or the Cray II, offer such capabilities, and both of these are priced far beyond the home enthusiast's budget. Although the ST does not have the capabilities of a dedicated graphics workstation, it is capable of displaying up to 256,000 pixels, for a fraction of the cost. Plus, the palette of 512 colors makes it ideal for producing colorful, exciting displays.

Only 16 colors may be displayed at one time (in low resolution). Each value stored in color memory corresponds to a hardware color register.

In GFA BASIC, the command COLOR defines the color to be used by a graphics command. If you do not specify a color before a graphics command, either the default color or the color last used is selected. Usually, the default color is color 1.

Graphics Programming Techniques

Entire books have been written about the various phases of computer graphics. Each section of this chapter could be dealt with in much greater detail. Instead, we're going to introduce you, quickly and painlessly, to graphics techniques, point you in the right direction, and then leave you to the excitement of discovery as you explore this world with your staunch ally, GFA BASIC.

As with any voyage, some preliminary attention to detail must be undertaken. After all, to use the graphics commands, you must at least be able to tell the computer where you want the object you're drawing to appear.

There are several coordinate systems in use, but the two most common are the Cartesian coordinate system and the polar coordinate system.

In the Cartesian coordinate system, the two-dimensional plane is assigned two axes: the horizontal axis, usually labeled *x*, and the vertical axis, usually labeled *y*. By convention, the positive direction of the *x*-axis is to the right, but the positive direction of the *y*-axis can be either up or down, depending on the application.

In geometry and most other mathematical applications, the positive direction of the *y*-axis is up, but on most computer systems, the positive direction is down. Points in the plane are represented by ordered pairs (*x,y*), where *x* is the distance of the point from the *y*-axis and *y* is the distance of the point from the *x*-axis. The point (0,0) is called the origin because it is the point where the *x* and *y* axes intersect, the point from which all other points in the plane are located.

The location of the origin is another disputed aspect of the Cartesian coordinate system. In geometry, the origin is traditionally located in the lower left corner of the display with the positive *x* and *y* axes extending toward the right and top of the display, respectively. However, on most computers, the origin is located in the upper left corner of the screen, because of the way the monitor draws the display.

Another possible location for the origin is the center of the display. This orientation makes it easy to graph both negative as well as positive values — something the other two systems ignore.

Either of these locations is equally valid, and conversion from one orientation to the other is simple, so the choice of which system to use is usually left up to the user.

Figure 3-1. Cartesian Coordinate System

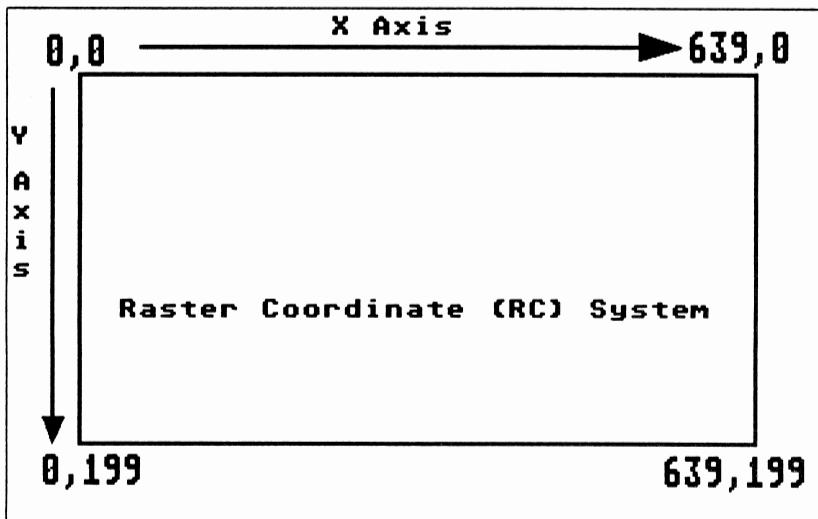
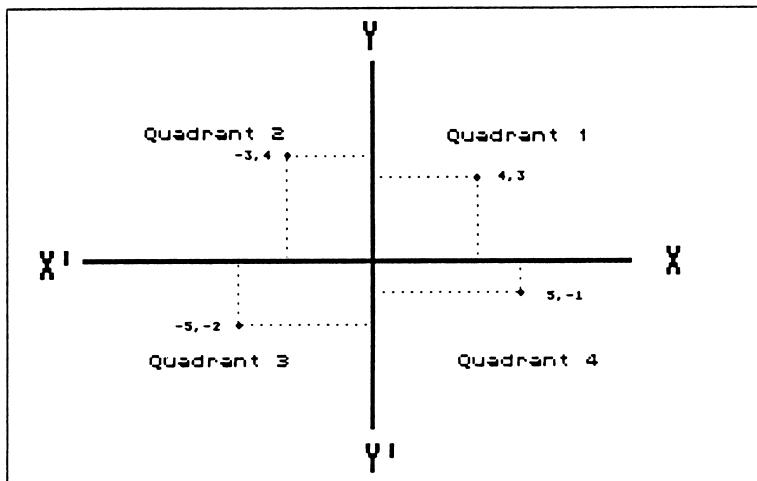
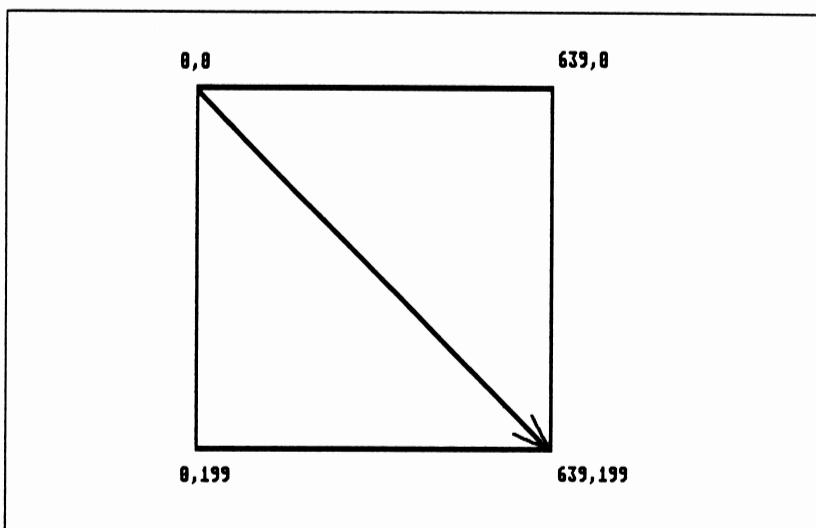


Figure 3-2 Polar Coordinate System

The Cartesian coordinate system corresponds closely with the real world, which makes it ideal for most applications, but in situations where rotation is necessary, the Cartesian coordinate system is difficult to use. Two systems of referencing are in common use with computers: the normalized device coordinate (NDC) system, and the raster coordinate (RC) system.

The NDC system addresses the graphics display independent of the device's display size, while the RC system addresses the device in actual display units. The RC system is the system used on most microcomputers. With this system, the screen is divided into rows and columns of dots, or pixels. The pixels in the top row have a vertical, or y, coordinate of 0. The y coordinate increases as you move toward the bottom of the screen. The bottom row of the screen has a y coordinate value of 199 on a color system and 399 on a monochrome system. The leftmost column of pixels has a horizontal, or x, coordinate of 0.

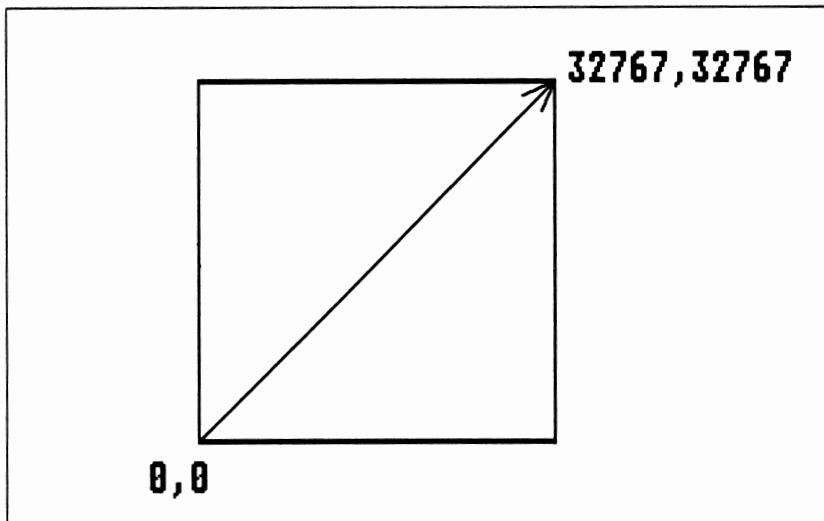
The x coordinate increases as you move toward the right edge of the display, where each pixel in the rightmost column has the value of 319 on color systems in low resolution and 639 in medium or high resolution.

Figure 3-3. Raster Coordinate System

The ST also supports the NDC system. It's difficult to write a single program that will work with the different types of devices, because almost every graphics output device has a different maximum horizontal and vertical resolution. That's where the NDC system becomes expedient.

The NDC system offers the programmer a means by which graphics drawn on one computer screen or printer will look the same when drawn on other computer screens or printers of different resolutions. With the NDC system, all graphics output is sent to an imaginary device that is 32,768 pixels wide and 32,768 pixels high. These pixels are grouped differently from those under the RC system. The y axis begins with 0 at the bottom of the screen, and moves up to the top row of pixels numbering 32,767. As in the RC system, the x axis is numbered from left to right. The leftmost column of pixels is 0, and the rightmost column is 32,767.

Two coordinate-referencing systems on one computer may seem difficult to grasp, but usually you'll only be dealing with the RC system in GFA BASIC. You should be aware of the NDC system, as it does have value with GDOS and some VDI functions.

Figure 3-4. Normalized Device Coordinate (NDC) System

Resolving a problem

The ST may be configured in low resolution (320x200 pixels) and medium resolution (640x200 pixels) with a color monitor, or in high resolution (640x400 pixels) with a monochrome monitor. It's very easy to include a simple check for the resolution mode, then use program modules applicable for the resolution. It's even possible to affect a change of resolution from GFA BASIC by using an XBIOS function. At the very least, you should include a check in your program to determine which mode of resolution the computer is displaying.

If necessary, you can then display an Alert box informing the user that the program requires a different resolution. Good programming practice dictates that your program be compatible with different systems, if possible. Especially with programs that use graphics, the effect you've labored to create will be lost if the proper mode of resolution isn't selected.

XBIOS is a group of extended input/output functions which are available to the programmer. For more information about XBIOS routines, refer to Appendix C. GFA BASIC makes these functions available to you by using the same bindings as those used by programmers working in C. To check the resolution of a system, XBIOS 4, called *getrez* in most literature about the ST, can be used.

```
PROCEDURE check_rez ..  
' ..  
rez=XBIOS(4) ..  
' ..  
RETURN ..
```

This procedure will return the screen resolution in the variable, *rez*:

0 = low resolution (320x200, 16 colors)
1 = medium resolution (640x200, 4 colors)
2 = high resolution (640x400, monochrome)

To change resolution modes, use the following procedure. Define *rez* equal to the value of the desired resolution then call *set_rez*. (Note: high resolution is only available with a monochrome monitor.)

```
PROCEDURE set_rez ..  
' ..  
dummy=XBIOS(5,L:-1,L:-1,W:rez) ..  
' ..  
RETURN ..
```

Switching from medium to low resolution will present no special problems. However, when you force a switch from low resolution to medium resolution, graphic commands will not appear on the right side of the screen. The operating system still assumes that values greater than 319 for X are off the screen. Meanwhile all graphics with X coordinates less than 320, will be shown, on the left side of the screen. Using the PRINT command to display text should present no problem.

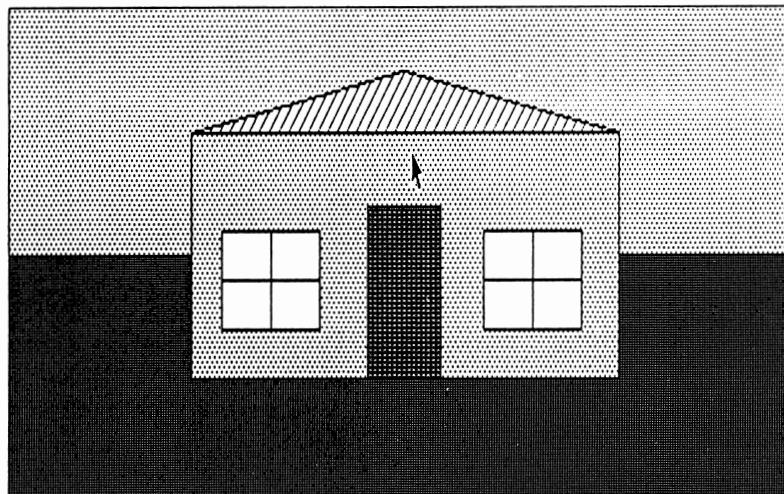
GFA BASIC Graphics Commands

GFA BASIC provides a powerful and easy to use programming environment. More information about these commands can be found in Chapter Two, GFA BASIC Commands and Functions.

Graphics on a 2-dimensional surface

The simplest of the graphic commands in GFA BASIC are the line drawing commands. The easiest of these is the LINE command. Remember, the coordinates used by the LINE command and the other GFA BASIC graphics commands are in the Raster Coordinate (RC) System.

Figure 3-5
Simple graphic produced by Program 3-1 using GFA BASIC.



The following short program demonstrates the ease with which a simple graphic screen can be developed using the LINE and BOX commands.

GFA BASIC listings for this book have been produced by first selecting the *Direct* item on the Editor's menu bar, then entering, DEFLIST 0, which causes all commands and function to be displayed (and printed LLIST) in all capital letters. Variable names and names of procedures will be in lower case letters.

Program 3-1 LINEDRAW.PRG

```
' Simple Line Drawing Demo ..  
' ..  
LINE 0,100,150,100 ..  
LINE 500,100,640,100 ..  
BOX 150,50,500,150 ..  
LINE 325,25,150,50 ..  
LINE 325,25,500,50 ..  
BOX 295,80,355,150 ..  
BOX 175,90,255,130 ..  
LINE 215,90,215,130 ..  
LINE 175,110,255,110 ..  
BOX 390,90,470,130 ..  
LINE 430,90,430,130 ..
```

```
LINE 390,110,470,110 ..
```

```
' ..
```

In order to make the screen a little more interesting, we'll use DEFFILL to select a pattern style and colors to be used by the FILL command. For more information about DEFFILL, or any other GFA BASIC command, refer to Chapter 2.

```
DEFFILL 3 ..
```

```
FILL 5,195 ..
```

```
' ..
```

```
DEFFILL 1,3 ..
```

```
FILL 330,30 ..
```

```
' ..
```

```
DEFFILL 1,2 ..
```

```
FILL 5,5 ..
```

```
' ..
```

```
DEFFILL 2,2 ..
```

```
FILL 155,55 ..
```

```
' ..
```

```
DEFFILL 2,1 ..
```

```
FILL 310,115 ..
```

The image produced will only flash on the screen, then disappear as the editor screen reappears when the program ends. We need to place GFA BASIC into a loop which repeats until we decide it's time to exit the loop (and the program) and send a signal to the interpreter. All that is needed is a REPEAT...UNTIL loop which will wait for any key to be pressed. When a key is pressed, the program ends.

```
' ..
```

```
REPEAT ..
```

```
UNTIL INKEY$<>" "
```

```
END ..
```

Generally, we'll be discussing each routine used in a program, rather than the individual commands used to obtain the desired result. If a more detailed explanation of a command or function is needed, refer to Chapter 2.

String Art

Only a short step from line drawing is an interesting art form, String Art. String Art may be either a succession of points connected by lines, or a sequence of lines drawn on the screen. Computer generated String Art may even take the form of lines dancing around the screen, as in this program.

STRING.BAS will draw each screen and then wait for the user to press a key before continuing. The last screen prompts the user for a positive number. This number determines how many petals to draw on a rose.

Originally we intended this part to work only with integers, but we discovered that some very interesting patterns develop if you enter compound numbers (numbers with a fractional part). Try several different values. STRING.BAS will display the rose for each and then wait for you to press a key. If you'd like to stop the drawing process at any time, press a key. When the drawing has stopped, pressing a key will prompt you for a new number. Enter 0 to exit STRING.BAS. The LINE command is used to draw each of the screens.

Program 3-2. String Art

The first thing String Art does is check the current screen resolution and set the global variables `x_size` and `y_size` to the width and height of the screen, respectively.

```
'What resolution is the ST in? ..  
'xbios(4) returns 0 if low , 1 if medium, and 2 if high resolution ..  
rez=XBIOS(4) ..  
' ..  
' set x_size to the width and y_size to the height of the screen ..  
' ..  
IF rez=0 ..  
    x_size=320 ..  
    y_size=200 ..  
ENDIF ..  
IF rez=1 ..  
    x_size=640 ..  
    y_size=200 ..  
ENDIF ..  
IF rez=2 ..  
    x_size=640 ..  
    y_size=400 ..  
ENDIF ..  
' ..
```

STRING.BAS is written so that each screen is drawn by a separate procedure; this makes it easy to delete some of the existing screens or add new ones of your own. After each screen is drawn, STRING.BAS waits for any key to be pressed before it clears the screen and draws the next screen.

```
GOSUB screen_1 ..  
GOSUB wait_key ..  
CLS ..  
GOSUB screen_4 ..  
GOSUB wait_key ..  
CLS ..  
GOSUB screen_2 ..  
GOSUB wait_key ..  
CLS ..  
GOSUB screen_3 ..  
GOSUB wait_key ..  
CLS ..  
GOSUB screen_5 ..  
' ..  
END ..
```

The rest of STRING.BAS consists of the procedures used in calculating or actually drawing the figures.

```
' ..  
' ..  
PROCEDURE wait_key ..  
' ..  
' This subroutine pauses the program until the user presses a key. ..  
' ..  
PROCEDURE wait_key ..  
REPEAT ..  
UNTIL inkey$<>"" ..  
RETURN !end of wait_key ..
```

The following procedure converts the radius and theta values of Polar coordinates to Cartesian coordinates. See the section on coordinate systems earlier in this chapter for details.

```
'  
'  
' PROCEDURE polar_to_rect '  
'  
' This procedure converts from the polar coordinate system to the '  
' rectangular coordinate system. Pass r (radius) and theta to '  
' polar_to_rect and it will return the x and y values in the '  
' GLOBAL VARIABLES x and y. '  
'  
PROCEDURE polar_to_rect(r,theta) '  
    x=r*COS(theta) '  
    y=r*SIN(theta) '  
RETURN ! end of polar_to_rect '  
'
```

This routine is used to draw two of the screens in STRING.BAS, by calculating the values necessary for the end points of the line to trace a limacon. Procedure limacon is used in the procedures screen_1 and screen_2. The first parameter of limacon is the angle theta for which the radius is to be calculated. The next two parameters are constants which modify the shape of the limacon. You might find it interesting to play around with these constants.

```
'  
'  
' PROCEDURE limacon '  
'  
' This procedure returns the value of r (radius) necessary to draw '  
' a limacon given the angle of theta and two scaling factors, m and n. '  
' NOTE: r is a GLOBAL VARIABLE. '  
'  
PROCEDURE limacon(theta,m,n) '  
    r=m+n*COS(theta) '  
RETURN ! end of limacon '  
'
```

Procedure rose generates one of the prettiest patterns possible using polar coordinates. Its output is used in Tscreen_5.

```
'  
'  
' PROCEDURE rose.  
'  
' This procedure returns the value of r (radius) necessary to draw.  
' a rose given the angle theta, the number of leaves, n, and a scaling.  
' factor m. n determines the number of leaves in the rose. If n is even,  
' then the rose will have 2n petals; if n is odd, there will be n leaves.  
' NOTE: r is a GLOBAL VARIABLE.  
'  
PROCEDURE rose(theta,n,m)  
    r=m*COS(n*theta)  
RETURN
```

End of rose denominator is used in screen_5 to determine how much of the flower to draw. The denominator returned by this procedure is used to calculate the upper limit of the function before it is plotted.

```
'  
'  
' PROCEDURE denominator.  
'  
' This routine returns the denominator of a number with a fractional component.  
'  
PROCEDURE denominator(num)  
    LOCAL k  
    LOCAL a  
    LOCAL b  
    '  
    k=1  
    found=0  
    a=num  
    GOSUB denom(a)  
    IF found=0  
        a=num/SQR(2)  
        GOSUB denom(a)  
    ENDIF
```

```
IF found=0 ..  
    a=num/SQR(3) ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    a=num/SQR(5) ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    a=num/PI ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    a=num/PI^2 ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    a=num/EXP(1) ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    a=num*PI ..  
    k=PI ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    a=num*PI^2^ k=PI^2 ..  
    GOSUB denom(a) ..  
ENDIF ..  
IF found=0 ..  
    k=1 ..  
ENDIF ..  
' ..  
d=b*k ..  
RETURN ..  
' ..  
' ..  
' PROCEDURE denom ..  
' ..  
' This routine actually calculates the denominator of the fraction. ..  
' ..
```

```
PROCEDURE denom(a) ↴
  LOCAL c ↴
  LOCAL dun ↴
  c=ABS(a) ↴
  b=1 ↴
  REPEAT ↴
    dun=0 ↴
    b=b/c ↴
    c=(1/c)-INT(1/c) ↴
    IF b>100000 ↴
      dun=1 ↴
    ENDIF ↴
  UNTIL c<1.0E-06 OR dun>0 ↴
  IF dun>1 ↴
    b=INT(b) ↴
    found=1 ↴
  ENDIF ↴
RETURN ↴
```

The remaining five procedures actually draw the screens. Each of them is well documented in the code so we won't go into detail on them.

```
' ↴
' ↴
' PROCEDURE screen_1 ↴
' ↴
' This procedure draws the first screen using the
procedures, ↴
' NOTE: the following variables are GLOBAL: rez, x_size, and y_size. ↴
' ↴
PROCEDURE screen_1 ↴
  LOCAL steps ↴
  LOCAL y_scale ly scaling factor ↴
  LOCAL x_scale lx scaling factor ↴
  LOCAL l_limit !lower limit of function ↴
  LOCAL u_limit !upper limit of function ↴
  LOCAL x_inc lx or theta increment ↴
  IF rez=0 ↴
    steps=1000*PI ↴
  ENDIF ↴
```

```
IF rez=1 ..  
    steps=2000*PI ..  
ENDIF ..  
IF rez=2 ..  
    steps=4000*PI ..  
ENDIF ..  
' ..  
y_scale=y_size/2 ..  
l_limit=0 ..  
u_limit=2*PI ..  
x_scale=x_size/2 ..  
x_inc=x_scale/steps ..  
' ..  
FOR theta=l_limit TO u_limit STEP x_inc ..  
    GOSUB limacon(theta,y_scale*3/4,y_scale*3/4) ..  
    GOSUB polar_to_rect(r,theta) ..  
    x1=x ..  
    y1=y ..  
    GOSUB limacon(theta,y_scale*3/4,-y_scale*3/4) ..  
    GOSUB polar_to_rect(r,theta) ..  
    x2=x ..  
    y2=y ..  
    IF rez<>1 ..  
        LINE x1+x_scale,y1+y_scale,x2+x_scale,y2+y_scale ..  
    ELSE ..  
        LINE x1*2+x_scale,y1+y_scale,x2*2+x_scale,y2+y_scale ..  
    ENDIF ..  
NEXT theta ..  
RETURN ! end of screen_1 ..  
' ..  
' ..  
' PROCEDURE screen_2 ..  
' ..  
' This procedure draws the second screen using the  
procedures, ..  
' NOTE: the following variables are GLOBAL: rez, x_size, and y_size. ..  
' ..
```

```
PROCEDURE screen_2 ..  
LOCAL steps ..  
LOCAL y_scale !y scaling factor ..  
LOCAL x_scale !x scaling factor ..  
LOCAL l_limit !lower limit of function ..  
LOCAL u_limit upper limit of function ..  
LOCAL x_inc !x or theta increment ..  
IF rez=0 ..  
    steps=250*PI ..  
ENDIF ..  
IF rez=1 ..  
    steps=500*PI ..  
ENDIF ..  
IF rez=2 ..  
    steps=1000*PI ..  
ENDIF ..  
' ..  
y_scale=y_size/2 ..  
l_limit=0 ..  
u_limit=2*PI ..  
x_scale=x_size/(u_limit-l_limit) ..  
x_inc=x_scale/steps ..  
'ferp ..  
FOR theta=l_limit TO u_limit STEP x_inc ..  
    GOSUB limacon(theta,y_scale*9/14,y_scale) ..  
    GOSUB polar_to_rect(r,theta) ..  
    x1=x ..  
    y1=y ..  
    GOSUB limacon(theta,y_scale*9/14,-y_scale) ..  
    GOSUB polar_to_rect(r,theta) ..  
    x2=x ..  
    y2=y ..  
    IF rez<>1 ..  
        LINE x2+x_size*15/16,y2+y_size/2,x1+x_size/16,y1+y_size/2 ..  
    ELSE ..  
        LINE x2+x_size*15/16,y2+y_size/2,x1*2+x_size/16,y1+y_size/2 ..  
    ENDIF ..  
NEXT theta ..  
RETURN ! end of screen_2 ..  
' ..  
' ..
```

```
' PROCEDURE screen_3 ..  
' ..  
' This procedure draws the third screen using the procedures, ..  
' NOTE: the following variables are GLOBAL: rez, x_size, and y_size. ..  
' ..  
PROCEDURE screen_3 ..  
LOCAL steps ..  
LOCAL y_scale !y scaling factor ..  
LOCAL x_scale !x scaling factor ..  
LOCAL l_limit !lower limit of function ..  
LOCAL u_limit !upper limit of function ..  
LOCAL x_inc !x or theta increment ..  
IF rez=0 ..  
    steps=500*PI ..  
ENDIF ..  
IF rez=1 ..  
    steps=750*PI ..  
ENDIF ..  
IF rez=2 ..  
    steps=1000*PI ..  
ENDIF ..  
' ..  
y_scale=y_size/2 ..  
l_limit=0 ..  
u_limit=8*PI ..  
x_scale=x_size/2 ..  
x_inc=x_scale/steps ..  
' ..  
FOR ctr%=l_limit TO u_limit STEP x_inc ..  
    x1=SIN(ctr%)+1 ..  
    x2=SIN(ctr%-3*PI/4)+1 ..  
    y1=COS(ctr%*3/4)+1 ..  
    y2=COS((ctr%-3*PI/4)*3/4)+1 ..  
    LINE x1*x_scale,y1*y_scale,x2*x_scale,y2*y_scale ..  
    PAUSE 1 ..  
NEXT ctr% ..  
RETURN ! end of screen_3 ..  
' ..  
' ..
```

```
' PROCEDURE screen_4 ..  
' ..  
' This procedure draws the fourth screen using the procedures, ..  
' NOTE: the following variables are GLOBAL: rez, x_size, and y_size. ..  
  
PROCEDURE screen_4 ..  
LOCAL steps ..  
LOCAL y_scale !y scaling factor ..  
LOCAL x_scale !x scaling factor ..  
LOCAL l_limit !lower limit of function^ LOCAL u_limit !upper limit of function ..  
LOCAL x_inc !x or theta increment ..  
IF rez=0 ..  
    steps=250*PI ..  
ENDIF ..  
IF rez=1 ..  
    steps=500*PI ..  
ENDIF ..  
IF rez=2 ..  
    steps=1000*PI ..  
ENDIF ..  
' ..  
y_scale=y_size/2 ..  
l_limit=0 ..  
u_limit=5*PI ..  
x_scale=x_size/2 ..  
x_inc=x_scale/steps ..  
' ..  
FOR ctr%=l_limit TO u_limit STEP x_inc ..  
    x1=SIN(ctr%)+1 ..  
    x2=SIN((ctr%-3*PI/4)+1 ..  
    y1=COS(ctr%*5/6)+1 ..  
    y2=COS((ctr%-3*PI/4)*5/6)+1 ..  
    LINE x1*x_scale,y1*y_scale,x2*x_scale,y2*y_scale ..  
    PAUSE 1 ..  
NEXT ctr% ..  
RETURN ! end of screen_4 ..  
' ..  
' ..
```

```
' PROCEDURE screen_5 ..J
' ..
' This procedure draws the fifth screen using the procedures, ..J
' NOTE: the following variables are GLOBAL: rez, x_size, and y_size. ..J
' ..

PROCEDURE screen_5 ..J
    LOCAL steps ..J
    LOCAL y_scale ly scaling factor ..J
    LOCAL x_scale lx scaling factor ..J
    LOCAL l_limit !lower limit of function ..J
    LOCAL u_limit !upper limit of function ..J
    LOCAL x_inc lx or theta increment ..J
    IF rez=0 ..J
        steps=500*PI ..J
    ENDIF ..J
    IF rez=1 ..J
        steps=1000*PI ..J
    ENDIF ..J
    IF rez=2 ..J
        steps=4000*PI ..J
    ENDIF ..J
    ..
    y_scale=y_size/2 ..J
    l_limit=0 ..J
    x_scale=x_size/2 ..J
    ..

PRINT " This routine draws flowers with the" ..J
PRINT "number of petals that you specify." ..J
PRINT "If you enter an odd number, n, a flower" ..J
PRINT "with n petals will be drawn. If you" ..J
PRINT "enter an even number, n, a flower with" ..J
PRINT "2n petals will be drawn. Flowers with" ..J
PRINT "large numbers of petals look like" ..J
PRINT "filled-in circles, so keep your number" ..J
PRINT "small. You can stop the drawing at any" ..J
PRINT "by pressing a key. When the drawing has" ..J
PRINT "stopped, press any key to continue." ..J
PRINT ..J
REPEAT ..J
    INPUT "Input a positive number (0 to quit) ";n ..J
    UNTIL n>=0 ..J
```

```
WHILE n>0 ..
    CLS ..
    u_limit=1 ..
    d=1 ..
    IF INT(n)<>n ..
        found=0 ..
        GOSUB denominator(n) ..
        u_limit=d ..
    ENDIF ..
    IF ODD(n*d) AND ODD(d) ..
        u_limit=PI*u_limit ..
    ELSE ..
        u_limit=2*PI*u_limit ..
    ENDIF ..
    steps=2000*PI*n ..
    x_inc=x_scale/steps ..
    theta=l_limit ..
    a$="" ..
    WHILE theta<=u_limit AND a$="" ..
        GOSUB rose(theta,n,y_scale) ! rose returns r ..
        GOSUB polar_to_rect(r,theta) ! polar returns x and y ..
        x1=x ..
        y1=y ..
        GOSUB rose(theta-PI/(3*n),n,y_scale) ..
        GOSUB polar_to_rect(r,theta-PI/(3*n)) ..
        IF rez=1 ..
            LINE x1*2+x_scale,y1+y_scale,x*2+x_scale,y+y_scale ..
        ELSE ..
            LINE x1+x_scale,y1+y_scale,x+x_scale,y+y_scale ..
        ENDIF ..
        theta=theta+x_inc ..
        a$=INKEY$ ..
    WEND ..
    GOSUB wait_key ..
    PRINT "Previous Number = ";n;" (";n*d;" / ";d;")" ..
    REPEAT ..
        INPUT "Input a positive integer (0 to quit) ";n ..
    UNTIL n>=0 ..
    WEND ..
RETURN ! end of screen_5 ..
```

Kaleidoscope

Here's a colorful demonstration program reminiscent of the Lava Lamps popular during the 1960s. You may have seen a version of this program running on the Commodore Amiga, or even a C language version on the Atari ST. KALEIDOSCOPE.BAS runs from the GFA BASIC interpreter.

This program also demonstrates the use of some new graphic commands. Note that "!" separates a comment from the command on the same line. Also, the ' mark is used in lines which contain only a comment.

In this program you'll see color spelled two ways; color and colour. Color is a GFA BASIC command which sets the color to be used by a graphics command, and the in-line Syntax checker will interpret it as such. Often to use a descriptive label or variable name, it's advantageous to deliberately mis-spell a word. Be sure to keep track of such mis-spellings, and mis-spell the word the same way every time.

Program 3-3. KALEDIO.BAS

Begin by reserving an area of memory to hold the alternate palette colors.

```
DIM palette(15),pal(15),pal2(15) !  
'!  
shape=0 ! Initial shape is a circle.  
HIDEM ! Hide mouse  
'!  
'!  
' Check for low resolution  
'!  
IF XBIOS(4)<>0  
alrt$="KALEIDOSCOPE only works in|Low Res (320X200 Pixels)"  
ALERT 3,alrt$,1,"Oops!!",b  
END  
ENDIF
```

Next, set up an alternate color palette. Each color is represented by three hexadecimal numbers, representing the three color guns of an RGB (Red, Green, and Blue) monitor, ranging from 0 for a gun which is turned off, to 7 for a gun at the highest output level. You can experiment with the numbers on the following table to observe the effects of different numbers on the display. (For more information about colors and the SETCOLOR command, refer to Chapter 2.)

You can use the Control Panel desk accessory to get instant feedback on mixing colors. When you use it to mix colors using different levels of red, green, and blue, the panel displays these color levels as numbers from 0 to 7. The desktop, and the Control Panel, will change colors as you change the settings. It's best to write down the numbers displayed as you proceed, as it may become necessary to turn off your ST and reboot in order to restore a readable screen. Use these numbers as the

hexadecimal values in the following table to install the colors in your color palette, as demonstrated below.

```
' ↴  
' ↴  
' Set up color table ↴  
' ↴  
pal2(0)=&H555 ↴  
pal2(1)=&H700 ↴  
pal2(2)=&H60 ↴  
pal2(3)=&H7 ↴  
pal2(4)=&H5 ↴  
pal2(5)=&H520 ↴  
pal2(6)=&H50 ↴  
pal2(7)=&H505 ↴  
pal2(8)=&H222 ↴  
pal2(9)=&H77 ↴  
pal2(10)=&H55 ↴  
pal2(11)=&H707 ↴  
pal2(12)=&H505 ↴  
pal2(13)=&H550 ↴  
pal2(14)=&H770 ↴  
pal2(15)=&H555 ↴  
' ↴
```

Now we'll display an Alert box containing the program title and a few pieces of necessary preliminary information. Usually, you won't want to use an Alert box for displaying anything except a warning, but in this case very little information is required.

Notice that as you type in the text for the variable *alrt\$* the screen will scroll to the left. When you move to the next line, a dollar sign is displayed at the point where the text exceeds the 80 character screen limit. This dollar sign signifies that more text exists on this line .

```
' ↴  
' Display title box ↴  
' ↴  
alrt$=" Kaleidoscope|Left Button Restarts|Right Button Changes Shape|Both Buttons End Program|" ↴  
ALERT 1,alrt$,1,"Okay",b ↴  
' ↴
```

Since we plan to tamper with the color scheme, it would only be polite to provide a means to reset the colors when we end our program.

```
GOSUB save_palette ↴  
' Setcolor 0,5,5,5 ! Set background to gray. ↴  
' ↴  
GOSUB set_colour ! Install new palette ↴  
' ↴  
start: ! Here's the real starting point ↴  
' ↴  
GOSUB restore_palette ! We want to restore colors for multiple runs ↴  
' otherwise print might be hard to read. ↴  
CLS ↴  
' ↴  
' Do some more start up stuff. ↴  
' ↴  
PRINT AT(1,10); ↴  
INPUT "How many pixels/step (1 to 10)";stp$ ↴  
IF stp$="" ↴  
    GOTO start ↴  
ENDIF ↴  
stp=VAL(stp$) ↴  
' ↴  
IF stp<1 OR stp>10 ! Check for valid range. ↴  
    GOTO start ↴  
ENDIF ↴  
' ↴  
minr=stp ↴  
CLS ↴  
FOR colour=1 TO 12 ↴  
    DEFTEXT colour,17 ↴  
    GOSUB title ↴  
NEXT colour ↴  
c=0 ↴
```

The next few lines determine the starting point for the first item to be drawn.

```
x=INT(RND(1)*320) .J
x1=x .J
y=INT(RND(1)*179)+20 .J
y1=y .J
r=minr .J
dr=1 .J
' .J
' Main program loop follows: .J
'REPEAT .J
' .J
GOSUB new_x .J
GOSUB new_y .J
' .J
DEFFILL c .J
INC c .J
IF c>15 .J
  c=1 .J
ENDIF .J
IF dr=1 .J
  INC r .J
ENDIF .J
IF dr=0 .J
  DEC r .J
ENDIF .J
IF r=20 .J
  dr=0 .J
ENDIF .J
IF r=minr .J
  dr=1 .J
ENDIF .J
' .J
```

Now determine which shape, circle or square to draw; then display that shape on the screen.

```
IF shape=0 ↴
    PCIRCLE x,y,r ↴
ELSE ↴
    PBOX x,y,x+r,y+r ↴
ENDIF ↴
' ↴
GOSUB change_colour ↴
' ↴
k=MOUSEK ↴
UNTIL k<>0 ↴
' ↴
' Left mouse button pressed? ↴
' ↴
IF k=1 ↴
    GOTO start ↴
ENDIF ↴
' ↴
' Right Mouse button pressed? ↴
' ↴
IF k=2 AND shape=0 ↴
    shape=1 ! Change shape to rectangle ↴
    GOTO start ↴
ENDIF ↴
' ↴
IF k=2 AND shape=1 ↴
    shape=0 ! Change shape to circle ↴
    GOTO start ↴
ENDIF ↴
' ↴
PAUSE 35 ! System needs a sec or 2 here. ↴
CLS ↴
GOSUB restore_palette ↴
SHOWM ↴
END ↴
' ↴
```

Now we need to find a new value for x.

```
'  
PROCEDURE new_x '  
'  
IF x1>=319 '  
SUB x,stp '  
ELSE '  
    ADD x,stp '  
ENDIF '  
ADD x1,stp '  
IF x1>=643 '  
    x1=stp^ x=stp '  
ENDIF '  
RETURN '  
'
```

Next we need a new y value.

```
'  
PROCEDURE new_y '  
'  
IF y1>199 '  
SUB y,stp '  
ELSE '  
    ADD y,stp '  
ENDIF '  
ADD y1,stp '  
IF y1>375 '  
    y1=28 '  
    y=28 '  
ENDIF '  
RETURN '  
'
```

The following procedure is a useful routine you'll see often in this book. It saves the palette as set up by the user from the Control Panel, so that the default colors may be restored when the program ends.

```
' Save Original Color Palette ..  
' ..  
PROCEDURE save_palette ..  
LOCAL i ..  
' ..  
FOR ctr% = 0 TO 15 ..  
    palette(ctr%) = XBIOS(7,W:ctr%,W:-1) ..  
    pal(ctr%) = palette(ctr%) ..  
NEXT ctr% ..  
' ..  
RETURN ..  
' ..
```

Here's the procedure which restores the default color palette. You'll see this one used again, too.

```
' ..  
' Restore Original Color Palette ..  
' ..  
PROCEDURE restore_palette ..  
LOCAL i ..  
' ..  
FOR ctr% = 0 TO 15 ..  
    SETCOLOR ctr%,palette(ctr%) ..  
NEXT ctr% ..  
RETURN ..  
' ..
```

To give the illusion of constant movement on the screen, we're going to rotate the color palette. This procedure takes care of the rotation sequence.

```
PROCEDURE change_colour ..  
' ..  
temp = pal(1) ..  
FOR ctr% = 2 TO 15 ..  
    pal(ctr%-1) = pal(ctr%) ..  
NEXT ctr% ..  
pal(15) = temp ..  
' ..
```

```
FOR ctr%=1 TO 15 ..  
    SETCOLOR ctr%,pal(ctr%) ..  
NEXT ctr% ..  
FOR ctr%=0 TO 15 ..  
    pal(ctr%)=XBIOS(7,W:ctr%,W:-1) ..  
NEXT ctr% ..  
RETURN ..  
' ..
```

Here is another routine to dazzle the beholder. This procedure sets up the title "Kaleidoscope" to alternate colors with color cycling.

```
PROCEDURE title ..  
IF colour=1 ..  
    TEXT 95,7,"K" ..  
ENDIF ..  
IF colour=2 ..  
    TEXT 105,7,"a" ..  
ENDIF ..  
IF colour=3 ..  
    TEXT 115,7,"l" ..  
ENDIF ..  
IF colour=4 ..  
    TEXT 125,7,"e" ..  
ENDIF ..  
IF colour=5 ..  
    TEXT 135,7,"i" ..  
ENDIF ..  
IF colour=6 ..  
    TEXT 145,7,"d" ..  
ENDIF ..  
IF colour=7 ..  
    TEXT 155,7,"o" ..  
ENDIF ..  
IF colour=8 ..  
    TEXT 165,7,"s" ..  
ENDIF ..  
IF colour=9 ..  
    TEXT 175,7,"c" ..  
ENDIF ..
```

```
IF colour=10 ..  
    TEXT 185,7,"o" ..  
ENDIF ..  
IF colour=11 ..  
    TEXT 195,7,"p" ..  
ENDIF ..  
IF colour=12 ..  
    TEXT 205,7,"e" ..  
ENDIF ..  
RETURN ..  
' ..
```

The set_color procedure is used to install our custom palette when the program initializes.

```
PROCEDURE set_colour  
FOR ctr% = 0 TO 15  
    dummy=XBIOS(7,pal2(ctr%))  
NEXT ctr%  
RETURN
```

The Dragon Plot

Fractals have been receiving a great deal of attention in mathematics and computer graphics. They're being used for everything from simulating random plant growth to generating realistic planetary landscapes in science fiction films.

Understanding fractals may not be as exciting as seeing them in action, but some explanation should prove helpful to your programming. The word fractal was coined by Benoit Mandelbrot, a pioneer in their study, to denote curves or surfaces having fractional dimension. The idea of fractional dimension can be illustrated as a straight curve (a line) which has only one-dimension, length. If the curve is infinitely long and curves in such a manner that it completely fills an area of the plane containing it, the curve can be considered two-dimensional. A curve partially filling an area has a fractional dimension between 1 and 2. Many types of fractals are self-similar. This means that all portions of the fractal resemble each other. This occurs whenever the whole is an expansion of some basic building block. In the language of fractals, this basic building block is called the generator.

The generator in the next program consists of a number of connected line segments. The curves plotted by this program are the result of starting with the generator and then repeatedly replacing each line segment, according to a defined rule. The number of cycles is limited by the screen resolution. Select too high a number of cycles and the program will also slow down to a crawl. Eventually, the

screen will be filled by the fractal generator, as portions are plotted off the screen as well.

This simple program which begins our exploration of the world of fractals plots a particular type of fractal which Mandelbrot labeled a "dragon plot". This program illustrates a self-contacting curve. A self-contacting curve touches, but does not cross, itself. The generator consists of two-line segments of equal length forming a right angle. During each cycle, the generator is substituted for each segment on alternating sides of the segments. (Even though GFA BASIC executes faster than most interpreted languages, this program is still slow, and that's part of the fascination of the dragon plot. Watch as your computer plots the figure, and note the areas of similarity.

DRAGON.BAS begins by asking you to enter an even number of cycles. When a plot is complete, pressing any key clears the screen and returns you to the starting prompt. Try starting out with two cycles, then four, then six, and so on. As you add more cycles, more time is required to fill in the dragon.

Program 3-4. DRAGON.BAS

```
DIM sn(20) ..  
in: ..  
CLS ..  
PRINT "Enter an even number of cycles (2-20)" ..  
INPUT "or enter a zero to quit: ";nc$ ..  
nc=VAL(nc$) ..  
IF nc=0 ..  
    END ..  
ENDIF ..  
IF nc<2 OR nc>20 ..  
    GOTO in ..  
ENDIF ..  
IF EVEN(nc)<>-1 ..  
    GOTO in ..  
ENDIF ..  
mk=128 ..  
FOR c=2 TO nc STEP 2 ..  
    mk=mk/2 ..  
NEXT c ..  
x=85 ..  
y=100 ..  
CLS ..  
COLOR 3 ..  
PLOT x,y ..
```

```
FOR c=0 TO nc ..  
    sn(c)=0 ..  
NEXT c ..  
rot_seg: ..  
d=0 ..  
FOR c=1 TO nc ..  
    IF sn(c-1)=sn(c) ..  
        d=d-1 ..  
        GOTO rotate_it ..  
    ENDIF ..  
    d=d+1 ..  
    rotate_it: ..  
    IF d=-1 ..  
        d=7 ..  
    ENDIF ..  
    IF d=8 ..  
        d=0 ..  
    ENDIF ..  
NEXT c ..  
IF d=0 ..  
    x=x+mk+mk ..  
    GOTO seg ..  
ENDIF ..  
IF d=2 ..  
    y=y+mk ..  
    GOTO seg ..  
ENDIF ..  
IF d=4 ..  
    x=x-mk-mk ..  
    GOTO seg ..  
ENDIF ..  
y=y-mk ..  
seg: ..  
COLOR 3 ..  
DRAW TO x,y ..  
sn(nc)=sn(nc)+1 ..  
FOR c=nc TO 1 STEP -1 ..  
    IF sn(c)>2 ..  
        c=1 ..  
        GOTO next_seg ..  
    ENDIF ..
```

```
sn(c)=0 ..  
sn(c-1)=sn(c-1)+1 ..  
next_seg: ..  
NEXT c ..  
IF sn(0)=0 ..  
    GOTO rot_seg ..  
ENDIF ..  
PRINT AT(7,23); "Press any key to continue" ..  
REPEAT ..  
UNTIL INKEY$<>"" ..  
GOTO in ..
```

The Mandelbrot Set

You can see fractals in nature in many places. The clouds, coastlines displayed in satellite photographs, and even a leaf will reveal fractals. We've seen one form of computer-generated fractals in Program 3-4, the dragon sweep. Now we'll explore one of the strangest and most beautiful areas of fractal geometry, the Mandelbrot set.

The Mandelbrot set consists of complex numbers, numbers with a *real* and *imaginary* part. These terms, real and imaginary, have historical significance in mathematics, but are no longer relevant. A complex number takes the form of $6 + 3i$; 6 is the real part of the number, and $3i$ represents the imaginary part (hence the i). Each complex number can be represented by a point on the two-dimensional plane.

The Mandelbrot set is located at the center of a two-dimensional sheet of numbers called the complex plane. When a specific operation is applied repeatedly to the numbers, the numbers outside the set retreat to infinity. The remaining numbers move about within the plane. Near the edge of the set, the numbers move about in a pattern which marks the beginning of the area of instability.

This area is astonishingly beautiful, complex, infinitely variable, and yet somehow, strangely similar. The unique factor in numbers within the Mandelbrot set is that values in the set never grow larger than 2 as the mathematical operation is performed on them. Points within the Mandelbrot set are represented in our program by black pixels. The colors of the other points are determined by counting the number of times each complex number is operated on before its value exceeds 2. This count is converted into a color.

For more information on the theory of fractals, see *The Fractal Geometry of Nature*, by Benoit Mandelbrot.

Figure 3-6. The Mandelbrot Primitive

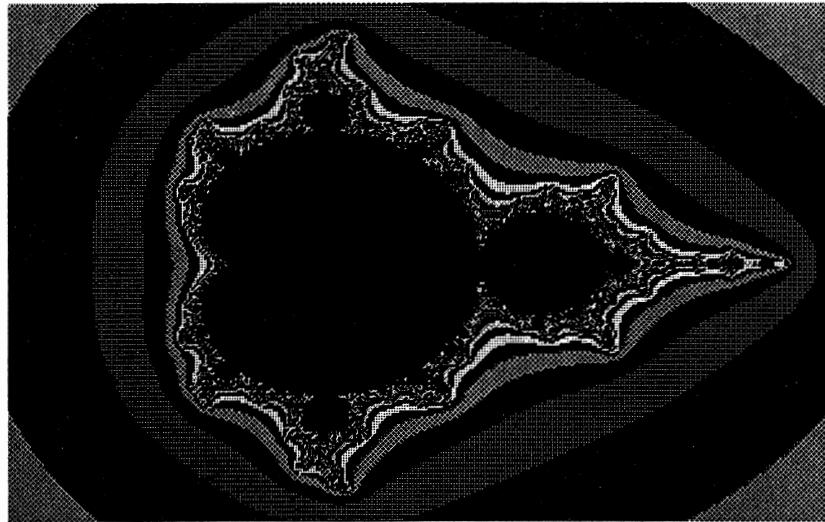
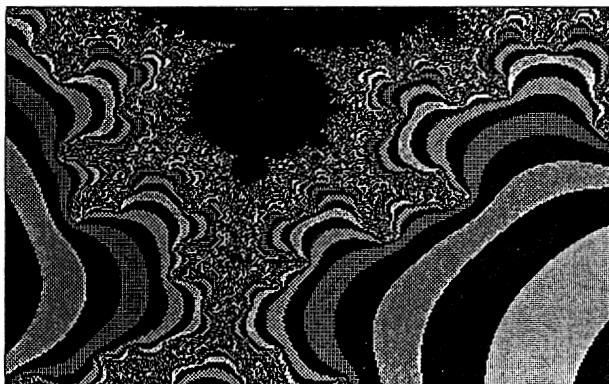


Figure 3-6 shows a representation of the entire Mandelbrot set. You can reproduce this figure by running MANDELBROT.BAS, and selecting 100 iterations, an X Center of .75, and a Y Center of 0. Choose a range of 2 for viewing the complete image.

Once you've become somewhat familiar with the general coordinates of the Mandelbrot set, try exploring the boundary area by selecting x and y coordinates located on the boundary, and then selecting smaller values for the range.

Program 3-5, MANDELBROT.BAS, was used to generate the images displayed in Figures 3-6, 3-7, and 3-8. Only a black-and-white representation was possible in the book, but the set generated by Program 3-5 is quite colorful. Some other interesting areas to explore follow.

Figure 3-7. Mandelbrot Figure



**Figure 3-8.
Another Segment of the Mandelbrot Boundary.**

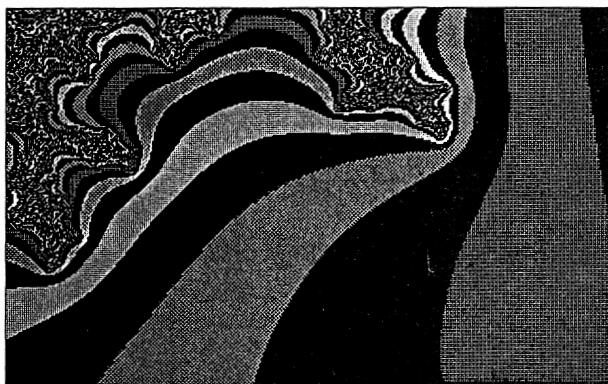
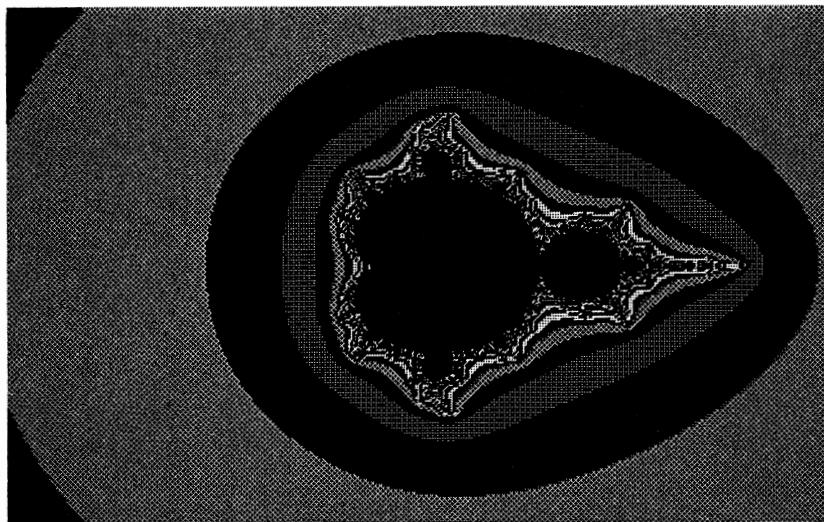


Figure 3-9
Another Mandelbrot Segment



<u>X-Center</u>	<u>Y-Center</u>	<u>Range</u>
-1	0.32	0.3
-1	0.2895	0.0005
-0.97	0.276	0.0001

In this program, we'll be introducing several new procedures which can be used in your own programs. We'll show you later how to merge these procedures into your programs.

Program 3-5. MANDELBROT.BAS

```
DIM palette(15),pal2(15) ↴
```

The mouse pointer will be in the way on the picture, so we'll just turn it off.

```
' ↴  
HIDEM ↴  
' ↴
```

We'll check to see which resolution is in use, then give a warning if the machine isn't in low resolution. This program will work in any resolution; it just looks best with 16 colors, so we'll allow it to continue if the user doesn't want to quit.

```
' ..
rez=XBIOS(4) ..
IF rez<>0 ..
  alrt$="Fractals look best in|Low Resolution" ..
  ALERT 1,alrt$,1,"Continue|Quit",b ..
  IF b=2 ..
    END ..
  ENDIF ..
ENDIF ..
'
```

Next we'll establish default values for different screen resolutions.

```
IF rez=0 ..
  screen_height=200 ! Low res screen ..
  screen_width=320 ..
ENDIF ..
'
IF rez=1 ! Medium resolution screen ..
  screen_height=200 ..
  screen_width=640 ..
ENDIF ..
'
IF rez=2 ! High resolution screen ..
  screen_height=400 ..
  screen_width=640 ..
ENDIF ..
'
' Main Loop begins here ..
'
```

We've used this routine before. It saves the palette so you can restore the original colors later. It's a good programming practice to clean up after yourself. If you change the palette, then exit to the GFA BASIC editor, you may find yourself stuck on a screen that is impossible to read.

```
GOSUB save_palette ..  
' ..
```

Here's the main loop which controls the program.

```
DO ..  
' ..  
GOSUB values ..  
IF rez=0 ..  
    GOSUB set_new_colors ..  
ENDIF ..  
GOSUB calculate ..  
IF rez=0 ..  
    GOSUB restore_palette ..  
ENDIF ..  
GOSUB display_it ..  
' ..  
LOOP ..
```

We'll put an END in here, but the program never really exits from the DO...LOOP. Other procedures are called and executed from procedures called by the DO...LOOP, but everything within this program actually occurs within this loop.

```
' ..  
END ..  
' ..
```

display_it is a procedure which displays information about the calculations for this particular portion of the Mandelbrot set and offers some options which may be selected by pressing a key. Usually this procedure will be encountered after the calculations for displaying the fractal are complete.

```

PROCEDURE display_it ..J
' ..
display: ..J
CLS ..J
PRINT AT(1,3); "Fractal Image" ..J
PRINT AT(1,5); count; " Iterations Calculated." ..J
seconds=INT((time_finish-time_start)/2)/100 ..J
min=INT(seconds/60) ..J
IF min<=0 ..J
  min=0 ..J
ENDIF ..J
seconds=ABS(INT((seconds MOD 60)*100)/100) ..J
PRINT AT(1,7); min; " Minutes and " ;seconds; " Seconds required." ..J
PRINT AT(1,8); "Number of iterations: " ;iteration_limit ..J
PRINT AT(1,10); "X Center: " ;x_center ..J
PRINT AT(1,11); "Y Center: " ;y_center ..J
PRINT AT(1,12); "Scaling factor: " ;range ..J
PRINT AT(1,14); "Minimum real value (xmin): " ;real_min ..J
PRINT AT(1,15); "Maximum real value (xmax): " ;real_max ..J
PRINT AT(1,16); "Minimum imaginary value (ymin): " ;imag_min ..J
PRINT AT(1,17); "Maximum Imaginary value (ymax): " ;imag_max ..J
PRINT AT(1,19); "Press: " ..J
PRINT AT(5,20); "<V> to View Fractal" ..J
PRINT AT(5,21); "<P> to Plot new Fractal" ..J
PRINT AT(5,22); "<S> to Save .NEO file" ..J
PRINT AT(5,23); "<Q> to Quit" ..J
' ..

```

key waits for the user to press a key. Then the key value is stored in *a\$*, and evaluated, and the proper branch is taken. UPPER is a GFA BASIC command which converts any value in *a\$* to uppercase. For more information about any GFA BASIC commands, refer to Chapter 2.

```

key: ..J
REPEAT ..J
  a$=INKEY$ ..J
  UNTIL a$<>""
' ..
IF UPPER$(a$)="V" ..J
  SGET screen2$ ..J
  IF rez=0 ..J
    GOSUB set_new_colors ..J

```

```
ENDIF ..  
SPUT screen1$ ..  
REPEAT ..  
UNTIL INKEY$<>"" ..  
GOSUB restore_palette ..  
SPUT screen2$ ..  
ENDIF ..  
' ..  
IF UPPER$(a$)="P" ..  
GOTO finito ..  
ENDIF ..  
' ..  
IF UPPER$(a$)="Q" ..  
SHOWM ..  
CLS ..  
END ..  
ENDIF ..  
IF UPPER$(a$)="S" ..  
' ..
```

Here's a handy procedure that saves the screen into a NEochrome-compatible file. Following this program, we'll give you a simple routine which can be merged with your programs, and show you how you can use this routine to load screens in your own programs.

```
IF rez<>0 CLS ..  
ALRT$="Must be Low Resfor Neochrome files" ..  
ALERT 1,alrt$,1,"OKAY",b ..  
ENDIF ..  
GOTO not_neo ..  
' ..  
SGET screen2$ ..  
default$="C:\*.NEO" ..  
FILESELECT default$,"",infile$ ..  
CLS ..  
IF infile$="" !equals "" if CANCEL selected ..  
GOTO not_neo ..  
ENDIF ..  
GOSUB set_new_colors ..  
SPUT screen1$ ..
```

```
OPEN "O",#1,infile$ ..  
' ..  
' First two bytes to zero ..  
' ..  
FOR ctr% = 0 TO 2 ..  
    OUT #1,0 ..  
    OUT #1,0 ..  
NEXT ctr% ..  
' ..  
' Save color palette ..  
' ..  
FOR ctr% = 0 TO 15 ..  
    hi=INT(pal2(ctr%)/256) ..  
    lo=pal2(ctr%)-256*hi ..  
    OUT #1,hi ..  
    OUT #1,lo ..  
NEXT ctr% ..  
' ..  
' Color cycling info (not needed) ..  
' ..  
FOR ctr% = 0 TO 89 ..  
    OUT #1,0 ..  
NEXT ctr% ..  
' ..  
' Save screen info ..  
' ..  
BPUT #1,XBIOS(3),32000 ..  
' ..  
CLOSE ..  
SPUT screen2$ ..  
GOSUB restore_palette ..  
' ..  
not_neo: ..  
' ..  
ENDIF ..  
' ..  
GOTO display ..  
' ..  
finito: ..  
RETURN ..  
' ..
```

values accepts input from the user for the values to be examined for the Mandelbrot boundaries.

```
PROCEDURE values ..  
' ..  
CLS ..  
INPUT "Number of iterations";iteration_limit ..  
INPUT "Center X";x_center ..  
INPUT "Center Y";y_center ..  
INPUT "Scale Range";range ..  
real_min=x_center-(range/2) ..  
real_max=x_center+(range/2) ..  
imag_max=y_center+((range/2)*0.77) ..  
imag_min=y_center-((range/2)*0.77) ..  
CLS ..  
RETURN ..  
' ..
```

The next procedure does the real work. The complex number is evaluated as defined. Several arbitrary values were selected for constants. You may find it interesting to change values, such as *max_limit*, to observe any changes in the figure plotted. *time_start* is used to calculate the time it took to complete the plot.

Note that even though GFA BASIC is one of the fastest interpreters available, fractals can take a lot of time to plot. Most designs will take over an hour to complete. The closer the complex numbers are to the Mandelbrot set, the more calculations are required before the result exceeds 2. Therefore, more time is required.

```
PROCEDURE calculate ..  
' ..  
time_start=INT(TIMER) ..  
max_limit=100000000 ..  
count=0 ..  
real=(real_max-real_min)/(screen_width-1) ..  
imaginary=(imag_max-imag_min)/(screen_height-1) ..  
FOR y=0 TO screen_height ..  
FOR x=0 TO screen_width ..  
lreal=real_min+x*real ..  
limag=imag_min+y*imaginary ..  
re=lreal ..  
im=limag ..
```

```
depth=0 ↴
calc_loop: ↴
INC count ↴
x1=re^2 ↴
y1=im^2 ↴
im=2*re*im-limag ↴
re=x1-y1-lreal ↴
INC depth ↴
IF ((depth=iteration_limit) OR ((x1+y1)>max_limit)) ↴
    GOTO finished_yet ↴
ELSE ↴
GOTO calc_loop ↴
ENDIF ↴
finished_yet: ↴
IF (depth<iteration_limit) ↴
    GOSUB draw_point ↴
ENDIF ↴
```

Since we can get stuck here waiting on the calculations for a long time, this will offer a way out and allow us a means to check on the time spent and calculations completed on this particular set of complex numbers.

```
IF INKEY$<>"" ↴
    GOSUB break ↴
ENDIF ↴
NEXT x ↴
NEXT y ↴
' ↴
' Save screen into screen1$ ↴
' ↴
SGET screen1$ ↴
' ↴
time_finish=INT(TIMER) ! time of completion ↴
' ↴
RETURN ↴
' ↴
```

If a key is pressed, we must stop to evaluate it. We'll also display an interim screen with some interesting information.

```
PROCEDURE break ..  
    ..  
    SGET screen1$ ..  
    time_finish=INT(TIMER) ..  
    IF rez=0 ..  
        GOSUB restore_palette ..  
    ENDIF ..  
    CLS ..  
    PRINT AT(1,3); "Fractal Image" ..  
    PRINT AT(1,5); count; " Iterations Calculated." ..  
    seconds=INT((time_finish-time_start)/2)/100 ..  
    min=INT(seconds/60) ..  
    IF min<=0 ..  
        min=0 ..  
    ENDIF ..  
    seconds=ABS(INT((seconds MOD 60)*100)/100) ..  
    PRINT AT(1,7); min; " Minutes and ";seconds; " Seconds required." ..  
    PRINT AT(1,8); "Number of iterations: ";iteration_limit ..  
    PRINT AT(1,10); "X Center: ";x_center ..  
    PRINT AT(1,11); "Y Center: ";y_center ..  
    PRINT AT(1,12); "Scaling factor: ";range ..  
    PRINT AT(1,14); "Minimum real value (xmin): ";real_min ..  
    PRINT AT(1,15); "Maximum real value (xmax): ";real_max ..  
    PRINT AT(1,16); "Minimum imaginary value (ymin): ";imag_min ..  
    PRINT AT(1,17); "Maximum Imaginary value (ymax): ";imag_max ..  
    PRINT AT(1,19); "Press:" ..  
    PRINT AT(5,20); "<C> to Continue Plot" ..  
    PRINT AT(5,21); "<E> to Return to Main Menu" ..  
    ..  
    keyit: ..  
    REPEAT ..  
        a$=INKEY$ ..  
    UNTIL a$<>" " ..  
    ..
```

```
IF UPPER$(a$)="E" ↴
  x=screen_width ↴
  y=screen_height ↴
  SPUT screen1$ ↴
  GOTO finit ↴
ENDIF ↴
' ↴
IF UPPER$(a$)="C" ↴
  IF rez=0 ↴
    GOSUB set_new_colors ↴
  ENDIF ↴
  SPUT screen1$ ↴
  GOTO finit ↴
ENDIF ↴
' ↴
GOTO keyit ↴
' ↴
finit: ↴
RETURN ↴
```

The calculations are completed; now it's time to plot the point. Depth is equal to the number of iterations calculated on the number which represents the point.

```
PROCEDURE draw_point ↴
' ↴
colour=depth MOD 16 ↴
COLOR colour ↴
IF (depth>100) ↴
  PLOT x,y ↴
ELSE ↴
  GOSUB greater_than_two ↴
ENDIF ↴
RETURN ↴
' ↴
```

```
PROCEDURE greater_than_two ↴
  ↴
  IF depth MOD 2 ↴
    PLOT x,y ↴
  ENDIF ↴
  RETURN ↴
  ↴
```

set_new_colors changes the color palette by installing the colors we've selected into the hardware register for the palette.

```
PROCEDURE set_new_colors ↴
  ↴
  ' Set up color table ↴
  ↴
  pal2(0)=&H0 ↴
  pal2(1)=&H75 ↴
  pal2(2)=&H765 ↴
  pal2(3)=&H401 ↴
  pal2(4)=&H410 ↴
  pal2(5)=&H530 ↴
  pal2(6)=&H241 ↴
  pal2(7)=&H62 ↴
  pal2(8)=&H474 ↴
  pal2(9)=&H744 ↴
  pal2(10)=&H731 ↴
  pal2(11)=&H165 ↴
  pal2(12)=&H14 ↴
  pal2(13)=&H750 ↴
  pal2(14)=&H423 ↴
  pal2(15)=&H601 ↴
  ↴
  ↴
FOR ctr%=0 TO 15 ↴
  SETCOLOR ctr%,pal2(ctr%) ↴
NEXT ctr% ↴
RETURN' ↴
```

We've seen the next two procedures before.

```
' Save Original Color Palette
' PROCEDURE save_palette
'   LOCAL ctr%
'   FOR ctr%=0 TO 15
'     palette(ctr%)=XBIOS(7,W:ctr%,W:-1)
'   NEXT ctr%
' RETURN

' Restore Original Color Palette
' PROCEDURE restore_palette
'   LOCAL ctr%
'   FOR ctr%=0 TO 15
'     SETCOLOR ctr%,palette(ctr%)
'   NEXT ctr%
' RETURN
```

Saving a Screen in NEOchrome™ Format

Here's a simple little routine similar to the one used in MANDELBROT.BAS. This program can easily be merged with any of your programs.

Program 3-6. NEOWRITE.BAS

```
PROCEDURE save_neoscreen ..  
'Reserve an area of memory for the alternate palette. ..  
' ..  
DIM pal2(15) ..  
' ..  
GOSUB save_palette ..  
HIDEM ..  
' ..  
SGET screen1$ ..  
' ..  
rez=XBIOS(4) ..  
IF rez<>0 ..  
    CLS ..  
    PRINT "Must be Low Resolution for Neochrome" ..  
    REPEAT ..  
    UNTIL INKEY$<>"" ..  
    CLS ..  
    SPUT screen1$ ..  
    GOTO Not_neo ..  
ENDIF ..  
' ..  
default$="A:\*.*.NEO" ..  
FILESELECT default$,"",infile$ ..  
CLS ..  
SPUT screen1$ ..  
OPEN "O",#1,infile$ ..  
' ..  
' First three words to zero ..  
' ..  
FOR ctr%=0 To 2 ..  
    OUT #1,0 ..  
    OUT #1,0 ..  
NEXT ctr% ..  
' ..  
' Save color palette ..  
' ..  
FOR ctr%=0 To 15 ..  
    hi=Int(pal2(ctr%)/256) ..
```

```
lo=pal2(ctr%)-256*hi ..J
OUT #1,hi ..J
OUT #1,lo ..J
NEXT ctr% ..J
' ..
' Color cycling info (not needed) ..J
' ..
FOR ctr%=0 To 89 ..J
    OUT #1,0 ..J
NEXT XTP% ..J
' ..
' Save screen info ..J
' ..
BPUT #1,XBIOS(3),32000 ..J
' ..
CLOSE ..J
' ..
not_neo: ..J
' ..
ENDIF ..J
' ..
' ..
finito: ..J
' ..
RETURN ..J
' ..
Save Original Color Palette ..J
' ..
PROCEDURE save_palette ..J
    LOCAL i ..J
    ' ..
    ' ..
    ' ..
FOR ctr%=0 TO 15 ..J
    pal2(ctr%)=XBIOS(7,W:ctr%,W:-1) ..J
NEXT ctr% ..J
RETURN ..J
' ..
```

After you've saved a screen in a NEOchrome™ format, you'll need a routine to display the screen without loading NEOCHROME.PRG™. Here's a routine to load any NEOchrome-format screen. With a minor alteration, you may load any screen into your programs without using the fileselector box.

Program 3-7. NEOLOAD.BAS

```
DIM palette(15),pal2(15) ↴
' ↴
GOSUB save_palette ↴
' ↴
```

Here's the main loop. Use these procedures as applicable in your program to load .NEO files. Refer to the user's guide for DEGAS Elite for information on how to handle a DEGAS file. It's just as simple.

```
main: ↴
' ↴
GOSUB check_rez ↴
GOSUB choose_pic ↴
HIDEM ↴
GOSUB load_colour ↴
GOSUB install_colour ↴
GOSUB show_pic ↴
' ↴
g$="" ↴
REPEAT ↴
  g$=INKEY$ ↴
UNTIL g$<>"" ↴
' ↴
CLS ↴
' ↴
GOSUB restore_palette ↴
' ↴
IF UPPER$(g$)="Q" ↴
  END
ENDIF ↴
' ↴
SHOWM ↴
```

```
GOTO main ↴
' ↴
' ↴
PROCEDURE check_rez ↴
' ↴
rez=XBIOS(4) ↴
IF rez<>0 ↴
  alrt$="NEOLOAD only works in|Low Resolution." ↴
  ALERT 3,alrt$,1,"Exit",b ↴
  END ↴
ENDIF ↴
' ↴
RETURN ↴
' ↴
PROCEDURE choose_pic ↴
' ↴
```

The next section was included to allow you to select a NEOchrome-format file using the standard GEM file selector. If you'd like to load a predetermined file, omit this procedure and define *filename\$* as the name of the desired program before calling Procedure Load_colour.

```
' ↴
disk$="*.NEO" ↴
FILESELECT disk$,"",filename$ ↴
IF filename$="" ↴
  END ↴
ENDIF ↴
' ↴
RETURN ↴
' ↴
PROCEDURE load_colour ↴
' ↴
OPEN "I",#1,filename$ ↴
temp$=INPUT$(38,#1) ↴
```

The first six bytes of the file don't matter, so we'll strip them off.

```
colour$=MID$(temp$,7,32) ..  
CLOSE #1 ..  
' ..
```

Next, we'll break down the two-byte color values into values which we can assign to our alternate palette, and install the colors.

```
palnum=0 ..  
count=0 ..  
REPEAT ..  
    hibyte=ASC(MID$(colour$,count,1)) ..  
    INC count ..  
    lobyte=ASC(MID$(colour$,count,1)) ..  
    INC count ..  
    pal2(palnum)=(hibyte*256)+lobyte ..  
    INC palnum ..  
UNTIL palnum=15 ..  
' ..  
RETURN ..  
' ..
```

The following XBIOS routine changes the hardware pointer to the palette to point to the variable Colour\$. Changing this pointer installs the new palette.

```
PROCEDURE install_colour ..  
' ..  
VOID XBIOS(6,L:VARPTR(colour$)) ..  
' ..  
RETURN ..  
' ..
```

Here's a little trick for loading the picture. XBIOS (2) is an XBIOS function which returns the location of the physical screen. This will vary for 520s and 1040s. Actually, we're cheating a bit here. We know that the first 128 bytes of a NEOchrome™ file contain information we're not interested in (information concerning the resolution, which is never used, the color palette, and the color rotation). What we'll do is ignore this information and begin loading the picture into an absolute memory location 128 bytes lower than the beginning of screen memory. This area isn't usually used for anything, and provides an easy way to strip off the unneeded bytes and load the picture image.

```
PROCEDURE show_pic ..  
  '..  
  physbase=XBIOS(2) ..  
  BLOAD filename$,physbase-128 ..  
  '..  
  RETURN ..  
  '..  
  ' Save Original Color Palette ..  
  '..  
PROCEDURE save_palette ..  
  '..  
  FOR ctr%=0 TO 15 ..  
    palette(ctr%)=XBIOS(7,W:ctr%,W:-1) ..  
  NEXT ctr% ..  
  '..  
  RETURN ..  
  '..  
PROCEDURE restore_palette ..  
  '..  
  FOR ctr%=0 TO 15 ..  
    SETCOLOR ctr%,palette(ctr%) ..  
  NEXT ctr% ..  
  '..  
  RETURN ..
```

Great Graphics

As you have seen from these simple examples, GFA BASIC makes graphics programming easy. It's not necessary to resort to PEEK's and POKE's to achieve the spectacular. This is only the starting point. As you continue to explore graphics with GFA BASIC, you'll be amazed at the magic you can create as you use the power of the 68000 Microprocessor.

In the next chapter, we'll introduce you to an even more impressive use of graphics — Animation. Soon you'll be writing the games you've dreamed about.

Chapter 4

Simple Animation



Chapter 4

Simple Animation

The foundations prepared in the previous chapters are sufficient, if put to practical use, to create impressive graphic displays. Anything from a simple line drawing to a 3-D display with hidden surface elimination is possible, using the simple commands and functions already discussed.

Computer animation is perhaps one of the most impressive features of any computer. However, even the power of the ST and the speed of GFA BASIC will disappoint you, unless you know a few little tricks.

This chapter will introduce you to a few of those tricks which you can use in your own animated displays. Many of the routines can be plugged into your own programs with ease. Although highly advanced graphic techniques are not discussed, you will be provided with the necessary background to continue to explore computer animation.

This chapter will also provide a handy tool, *Shape Editor*, for designing your own graphics images and then merging them into your programs.

The Animation Sequence

Animation is the process of creating a dynamic display of images by providing image elements which appear to have motion. Animated graphics are used in simulations such as flight simulators, and of course, the arcade action games.

The objective of computer animation is to generate a series of images which appear to move smoothly and naturally, without compromising speed.

Television and movies have been using animation for years. The technique they employ is to create a series of pictures with minute changes between them, then display the sequence at rates of over 15 frames per second.

This technique is known as full-frame animation, and is ideal for movies and television, where the hardware supports the application. However, in computer terms, full frame animation requires a tremendous amount of data to be passed. This is difficult enough, even when using the analog technology of the film industry. Recomputing total screens with a microprocessor is simply out of the question.

However, computer animation uses a process of updating the screen information *only* for the area of the screen which has undergone a change. Instead of refreshing an entire screen, only a few thousand pixels need to be updated.

Even from within the GFA Editor/Interpreter, GFA BASIC is equal to the challenge. Visual displays, which only a few years ago would have been thought impossible, are easily within reach of all programmers.

Screen Flicker

In computer animation and graphics, one of the most serious problems to be dealt with is flicker.

Flicker is (usually) an undesirable effect and is best described as a blinking of the display caused by the blank period between the time the screen is erased, and the time a new screen is displayed.

Flicker is really not related to the smoothness of the animation, but can distract the eye enough to spoil even the most carefully crafted animation. Fortunately, GFA BASIC provides a handy tool for dealing with flicker, the VSYNC command.

Quite simply, VSYNC causes the computer to wait until the optimum instant before refreshing the screen.

Page Flipping

One effect method of reducing the flicker of the screen is to utilize a technique known as "double buffering" or "page flipping".

This method is really deceptively simple, just have more than one screen in memory, and rapidly switch the display between them.

Generally, the image which is being displayed is a static, or unmoving, image, and all changes and drawing are done on a "hidden" screen. When the image is completed, the screens are "flipped", and the new image is displayed. Program 4-1, Bouncing Balls, uses this technique.

First Steps

At it's most simplistic level, animation is deceiving the eye into seeing continuous motion as a series of still images are viewed. This technique is the same whether the animation is an animated cartoon for TV or the movies, or a computer generated image.

As a matter of fact, many of the cartoons you see on television are generated from computer images.

Let's start out with a very simple illustration, but one that uses the techniques necessary for advanced computer animation.

Bouncing Balls

This simple program animates two balls, bouncing around the screen. Although quite simple, advanced techniques, such as page flipping are illustrated.

Program 4-1 Bouncing Balls

```
'Bouncing Ball Demo ..J
DIM s%(32255/4)           !Reserve space for an alternate screen ..J
```

Let's start out by finding out which resolution mode the program is running under. By a rather simple process of assigning values to variables, we can make sure that all three resolutions are supported.

Usually you won't have to worry about this, other than to prevent your program from running in a resolution not intended, as was demonstrated in Chapter 3, Graphics.

When the resolution has been determined with a call to the XBIOS(4) function, values may be assigned for the x and y boundaries limiting movement of the balls.

```
rez%=XBIOS(4) ..J
IF rez%>0                 !Is it low rez? ..J
  h%=275                   !Allow room for the ball to stay on the screen ..J
  v%=160 ..J
ENDIF ..J
IF rez%>1                 !Is it medium rez? ..J
  h%=595                   !Allow room for the ball to stay on the screen ..J
  v%=160 ..J
ENDIF ..J
IF rez%>2                 !Is it high rez? ..J
  h%=595                   !Allow room for the ball to stay on the screen ..J
  v%=380 ..J
ENDIF ..J
rate%=2                     !Variable for speed of movement ..J
DEFFILL 2                  !Set fill color for drawing ..J
GRAPHMODE 2                 !Set mode for drawing as transparent ..J
```

You might want to change the value for GRAPHMODE to other values for experimenting.

```
SETCOLOR 0,3,4,5            !Define a new color. ..J
SETCOLOR 15,7,7,7           !Define a new color ..J
```

If you really want to get fancy, use the Save_color routine from Chapter 3 to save the default desktop, then use the Restore_palette routine (also from Chapter 3) when the program ends. For simplicity, just set the palette to readable colors (black and white) when ending.

PCIRCLE 100,100,20	!Draw a solid circle on the screen ..
GET 75,75,125,125,a\$!Capture the circle into a\$..
CLS	!and clear the screen ..
DEFFILL 3	!Set a new color for drawing ..
PCIRCLE 100,100,20	!Draw another circle ..
GET 75,75,125,125,b\$!Save it as b\$..
CLS ..	

Now, do some necessary preliminary functions to get set up for the animation sequences.

a%=XBIOS(3)	!Get the screen logical base address ..
b% = VARPTR(s%(0))+255 AND &HFFFF00	!pointer to alternate screen ..
SGET h\$!Save screen ..

Define a starting location for the two balls. x% and y% are the coordinates for the first ball. bx% and by% are the coordinates for the second ball.

x%=100 ..
y%=100 ..
bx%=0 ..
by%=0 ..

Overhead done! Now it's time to put things in motion.

REPEAT ..	
VSYNC	!Wait for vsync (minimize flicker) ..
SWAP a%,b%	!get ready to flip screens ..
VOID XBIOS(5,L:a%,L:b%,-1)	!Flip screens now ..
VSYNC ..	
SPUT h\$!Restore background and effectively erase old image. ..

What's happening is that the background (h\$) is placed on the alternate screen, which isn't displayed yet, providing a clean slate to draw the balls at a new location, as the following section of code takes care of.

This routine is pretty straight forward. Just check the x,y coordinates of both objects, compare to the preset maximum and minimum values, then add or subtract the movement rate (rate%) as necessary.

```
IF xflag!=FALSE ..  
    ADD x%,rate% ..  
ELSE ..  
    SUB x%,rate% ..  
ENDIF ..  
IF yflag!=FALSE ..  
    ADD y%,rate% ..  
ELSE ..  
    SUB y%,rate% ..  
ENDIF ..  
IF x%>h% ..  
    xflag!=TRUE ..  
ENDIF ..  
IF x%<=0 ..  
    xflag!=FALSE ..  
ENDIF ..  
IF y%>v% ..  
    yflag!=TRUE ..  
ENDIF ..  
IF y%<=-10 ..  
    yflag!=FALSE ..  
ENDIF ..  
IF bxflag!=FALSE ..  
    ADD bx%,rate% ..  
ELSE ..  
    SUB bx%,rate% ..  
ENDIF ..  
IF byflag!=FALSE ..  
    ADD by%,rate% ..  
ELSE ..  
    SUB by%,rate% ..  
ENDIF ..
```

```
IF bx%>h% ..  
    bxflag!=TRUE ..  
ENDIF ..  
IF bx%<0 ..  
    bxflag!=FALSE ..  
ENDIF ..  
IF by%>v% ..  
    byflag!=TRUE ..  
ENDIF ..  
IF by%<10 ..  
    byflag!=FALSE ..  
ENDIF ..
```

Put the two balls on the alternate screen, then loop back to display them.

```
PUT x%,y%,a$,7 ..  
PUT bx%,by%,b$,7 ..  
UNTIL MOUSEK ..
```

If a mouse button was pressed, the program ends, but first cleans up after itself by resetting the screen pointer, and setting black and white colors so the screen can be read easily.

```
a%=MAX(a%,b%) ..  
VOID XBIOS(5,L:a%,La%,-1) ..  
SETCOLOR 0,7,7,7 ..  
SETCOLOR 15,0,0,0 ..
```

At it's most simplistic level, this is all that's required for animation. Sometimes intricate patterns of movement are required, and the calculation of these locations can become time consuming, even for a computer.

The next example demonstrates a method of storing the necessary information in any array. Of course this method only works for repeating patterns.

Orbit

This example places two objects in orbit around a central object. Although designed to run in low resolution, the effect can be observed in all resolution modes.

This program is actually less complicated than the bouncing balls of Program 4-1. All we're doing is plotting a circle on the screen, then erasing it and drawing a new circle.

It's very simple, but effective in producing the illusion of motion on a simple background.

Program 4-2 Orbit

Start the program by dimensioning space to hold the position coordinates of the orbiting objects. In this case, I've chosen to use 200 locations for plotting.

Since the mouse isn't used, just hide it.

```

DIM x%(200),y%(200),x1%(200),y1%(200) ...
HIDEM                                     !Mouse isn't used, so hide it ..
SETCOLOR 0,0,0,0                           !Set background to black ..
SETCOLOR 15,7,7,7                           !Set foreground (text) to white ..
CLS ..
GET 1,1,30,30,a$                          !Get a block of background ..
' Define orbit of object 1 ..
xc=155 ..
yc=90 ..
yr=150 ..
xr=80 ..
' This routines plots an elliptical orbit ..
FOR ctr%=0 TO 2*PI STEP PI/100 ..
  x%(ctr%)=INT(SIN(ctr%)*yr+xc) ..
  y%(ctr%)=INT(COS(ctr%)*xr+yc) ..
  INC c% ..
NEXT ctr% ..
' Define orbit for object 2 ..
c%=0 ..
xc=135 ..
yc=85 ..
yr=115 ..
xr=50 ..

```

```

FOR ctr%=2*PI TO 0 STEP -PI/100 ..J
  x1%(c%)=INT(SIN(ctr%)*xr+xc) ..J
  y1%(c%)=INT(COS(ctr%)*xr+yc) ..J
  INC c% ..J
NEXT ctr% ..J
' Now prepare for animation ..J
c%=1                      !starting value for counter ..J
DEFFILL 3                  !Fill central (unmoving) object
PCIRCLE 160,100,10          !Fill color for satellites
DEFFILL 2

```

Now put things in motion. First stamp the block of the background (a\$) onto the screen, then plot a new object for each satellite.

```

' ..J
' Animation sequence ..J
' ..J
REPEAT ..J
  PUT x%*(c%-1)-11,y%*(c%-1)-10,a$ ..J
  PCIRCLE x%(c%),y%(c%),5 ..J
  PUT x1%*(c%-1)-11,y1%*(c%-1)-10,a$ ..J
  PCIRCLE x1%(c%),y1%(c%),5 ..J

```

The variable c% is a counter which points to the position, as stored in the array, for the object to be displayed.

```

INC c% ..J
IF c%>200 ..J
  c%=1 ..J
ENDIF ..J
UNTIL MOUSEK               !Exit loop if mouse button is pressed ..J
SHOWM                      !Restore mouse ..J
SETCOLOR 0,7,7,7             !Set background to white ..J

```

Values other than one may be used for c%, starting the objects from other locations on the screen. I've used this technique in several arcade type programs.

Shape Editor

In order to fully and easily utilize the graphics potential of GFA BASIC, some tools are handy. Shape Editor is a full featured GEM based program which permits the user to design 16 by 16 pixel objects. This program works only in low resolution.

As an extra feature, screen colors may be modified easily, and saved in a format which is easily installed in any GFA BASIC low resolution program.

The shapes may be saved as .LST files, or as .SHP files. .LST files may be merged with GFA BASIC programs, while .SHP files may be loaded into string variables, in the proper format to be used by the PUT command.

Since this is a full featured GEM program, and the first in this book, an explanation will be given for each program segment.

Program 4-3 Shape Editor

Some overhead is required for storage of variables, so begin by DIMing an area to hold these values.

As usual, the second step is to use the save_palette routine, introduced in Chapter 3, to save the default palette.

```
DIM palette%(15),strip$(29),alrt$(7),pic$(3) ..  
@save_palette ..
```

An array of variables, strip\$(), holds all the information for the menu bar. This information can be stored in a data statement, then read into a variable when needed during the initialization process.

```
DATA Desk , About Object Editor ..  
DATA ----- ..  
DATA ----,"" ..  
DATA Options , Set Colors , Clear Shape , Save Shape , Load Shape ..  
DATA Save Palette , Load Palette , Quit,"" ..  
DATA Picture , Picture 1 , Picture 2 , Picture 3 , Picture 4 ..  
DATA -----, Animate ,"" ..  
DATA xxx ..  
DATA " |Exit Object Editor || Are you sure?" ..  
DATA " GFA Object Editor | By George Miller| Copyright 1988|All Rights Reserved" ..  
DATA " Save Shape || Save File as:" ..  
DATA " |Save Palette as:" ..  
DATA " |This is not a|Color File!" ..  
DATA "This program works|in Low Resolution| Only!" ..
```

```
DATA "Animate" | |How many images?" ↴  
DATA "xxx" ↴
```

This menu will permit the use of Desk Accessories, as specified in the third line of data statements. Unfortunately, this version of GFA BASIC (Version 2.xx) doesn't really handle desk accessories very well. A "hole" will remain in the screen after a desk accessory is closed.

The next series of data statements will be read into an array called alrt\$(). These will be displayed by the ALERT function as needed by the program.

In theory, all these data statements should have been planned before the first line of program code was written. In practice, however, they are usually added as the program as is developed.

The "xxx" at the end of each set permits the data to be read from a loop, which will exit if the string "xxx" is found.

If you use this technique, remember to change the size of the array to reflect the number of elements which are to be read.

This next sequence reads the data statements, builds the array used by the menu, and sets up the variable array, pic\$(0), for holding the shapes for animation later.

```
FOR pc%=0 TO 3 ↴  
    GET 251,41,268,58,pic$(pc%) ↴  
NEXT pc% ↴  
blank$=pic$(0) ↴  
ctr%=0 ↴  
DO ↴  
    READ strip$(ctr%) ↴  
    EXIT IF strip$(ctr%)="xxx" ↴  
    INC ctr% ↴  
LOOP ↴  
strip$(ctr%)="" ↴  
strip$(ctr%+1)="" ↴
```

Now read the data for the array used in the alrt\$(0) array.

```
ctr% = 0 ..  
pc% = 0 ..  
DO ..  
    READ alrt$(ctr%) ..  
    EXIT IF alrt$(ctr%) = "xxx" ..  
    INC ctr% ..  
LOOP ..  
' ..
```

Check the resolution, this program was created for low resolution only. If anything besides low resolution is found, show an alert box informing the user, and abort the program.

```
rez% = XBIOS(4) ..  
IF rez% <> 0 ..  
    ALERT 3, alrt$(5), 1, "OK", b ..  
    END ..  
ENDIF ..
```

If everything is ok, continue with the initialization sequence.

```
DO ..  
    @initialize ..  
    @activate_menu ..  
    MENU 20+pc%, 1 ..
```

Nest two loops, because later a flag (n!) will be set to erase all data, and start the program with a clean screen.

The inner loop actually runs the program. ON MENU will respond to any menu selections made by the user.

```
DO ↴  
    EXIT IF n!=TRUE ↴  
    ON MENU ↴  
    LOOP ↴  
    n!=FALSE ↴  
LOOP ↴  
' ↴
```

If a menu message is received, Procedure menu takes care of it, calling additional subroutines as needed.

Be careful to type in the menu headings *exactly* as listed in the data statements, or the IF statements may not find a match for a menu selection. All of the following phrases inside the quotation marks are preceded by two spaces, and most are followed by two spaces. Where no trailing spaces are used, a comment will inform you.

```
PROCEDURE menu ↴  
    IF strip$(MENU(0))=" About Object Editor"           !No trailing spaces ↴  
        ALERT 0,alrt$(1),1," OK ",b ↴  
    ENDIF ↴  
    IF strip$(MENU(0))=" Quit"                          !No trailing spaces ↴  
        ALERT 2,alrt$(0),1,"Exit|Stay",b ↴  
        IF b=1 ↴  
            @restore_palette ↴  
            SHOWM ↴  
            MENU KILL ↴  
            END ↴  
        ENDIF ↴  
    ENDIF ↴  
    IF strip$(MENU(0))=" Clear Shape " ↴  
        pic$(pc%)=blank$ ↴  
        n!=TRUE ↴  
    ENDIF ↴  
    IF strip$(MENU(0))=" Save Shape " ↴  
        @sav_it ↴  
    ENDIF ↴  
    IF strip$(MENU(0))=" Load Shape"                   !No trailing spaces ↴  
        @load_it ↴  
    ENDIF ↴  
    IF strip$(MENU(0))=" Set Colors " ↴  
        @set_colors ↴
```

```
ENDIF ..  
IF strip$(MENU(0))=" Save Palette " ..  
    @store_palette ..  
ENDIF ..  
IF strip$(MENU(0))=" Load Palette " ..  
    @load_palette ..  
ENDIF ..  
IF strip$(MENU(0))=" Picture 1 " ..  
    @clear_pic_num ..  
    pc%=0 ..  
    @set_check ..  
ENDIF ..  
IF strip$(MENU(0))=" Picture 2 " ..  
    @clear_pic_num ..  
    pc%=1 ..  
    @set_check ..  
ENDIF ..  
IF strip$(MENU(0))=" Picture 3 " ..  
    @clear_pic_num ..  
    pc%=2 ..  
    @set_check ..  
ENDIF ..  
IF strip$(MENU(0))=" Picture 4 " ..  
    @clear_pic_num ..  
    pc%=3 ..  
    @set_check ..  
ENDIF ..  
IF strip$(MENU(0))=" Animate " ..  
    MENU OFF ..  
    @clear_pic_num ..  
    DEFTEXT 2 ..  
    TEXT 235,68,"Animate" ..  
    @animate ..  
    DEFTEXT 0 ..  
    TEXT 235,68,"Animate" ..  
    DEFTEXT 1 ..  
    !No trailing spaces ..
```

```
ENDIF ↴
MENU OFF ↴
RETURN ↴
' ↴
```

Now display a line to inform the user which picture (of the 4 frames available) is being displayed.

```
PROCEDURE clear_pic_num ↴
DEFTEXT 0 ↴
TEXT 220,80,"Picture "+STR$(pc%+1) ↴
DEFTEXT 1 ↴
RETURN ↴
' ↴
```

Procedure set_check puts a checkmark on the menu at the appropriate position.

```
PROCEDURE set_check ↴
LOCAL ctr% ↴
FOR ctr%=20 TO 24 ↴
    MENU ctr%,0 ↴
NEXT ctr% ↴
MENU 20+pc%,1 ↴
pic$=pic$(pc%) ↴
DEFMOUSE 1 ↴
@new_pic ↴
TEXT 220,80,"Picture "+STR$(pc%+1) ↴
DEFMOUSE 0 ↴
RETURN ↴
'
```

Here's the save_palette routine.

```
PROCEDURE save_palette ..  
LOCAL ctr% ..  
FOR ctr%=0 TO 15 ..  
    palette%(ctr%)=XBIOS(7,W:ctr%,W:-1) ..  
NEXT ctr% ..  
RETURN ..  
' ..
```

Since we saved the palette, we need a routine to restore it. This routine was also introduced in Chapter 3.

```
PROCEDURE restore_palette ..  
SHOWM ..  
LOCAL ctr% ..  
FOR ctr%=0 TO 15 ..  
    SETCOLOR ctr%,palette%(ctr%) ..  
NEXT ctr% ..  
RETURN ..  
' ..
```

This procedure takes care of the actual initialization process.
The menu is initialized and the screen is prepared.

```
PROCEDURE initialize ..  
CLS ..  
COLOR 1 ..  
MENU strip$() ..  
GRAPHMODE 2 ..  
DEFLINE 1,3 ..  
BOX 17,19,182,181 ..  
DEFLINE 1,1 ..  
PLOT 15,18      !Clean up box ..  
PLOT 16,18      !Clean up box ..  
' draw grid ..  
FOR ctr%=0 TO 16 ..  
    DRAW 20,ctr%*10+20 TO 181,ctr%*10+20 ..  
    DRAW 20+ctr%*10,20 TO 20+ctr%*10,180 ..  
NEXT ctr% ..
```

The next segment draws a series of boxes, which display the color palette available to be used.

```
' set up palette ..J
FOR ctr% = 0 TO 15 ..J
    DEFFILL 1 ..J
    BOX 20+ctr%*10,184,30+ctr%*10,196 ..J
    DEFFILL ctr%,1 ..J
    PBOX 21+ctr%*10,185,29+ctr%*10,195 ..J
NEXT ctr% ..J
```

Now, let's put up a simple title box.

```
DEFFILL 1 ..J
PBOX 206,150,319,199 ..J
DEFTEXT 3,1,,6 ..J
TEXT 226,160,"GFA BASIC" ..J
TEXT 210,170,"Object Editor" ..J
DEFTEXT 0,1,,4 ..J
TEXT 215,180,"By George Miller" ..J
TEXT 220,190,"Copyright 1988" ..J
BOX 250,40,269,59 ..J
```

The variable c% contains the drawing color. The box drawn now reminds the user what color for the pen has been selected.

```
c%=1      !initialize drawing color ..J
PBOX 210,100,230,120 ..J
DEFTEXT 1,1,,6 ..J
TEXT 240,117,"Pen Color" ..J
RETURN ..J
' ..J
```

Procedure activate_menu defines a box which will be used to call Procedure new_color. If the user moves the mouse into the palette display area, a new drawing color may be selected.

```
PROCEDURE new_color ..  
IF MOUSEK=1 ..  
x%=MOUSEX ..  
y%=MOUSEY ..  
c%=POINT(x%,y%) ..  
DEFFILL c% ..  
COLOR c% ..  
PBOX 210,100,230,120 ..  
ENDIF ..  
RETURN ..  
' ..
```

Procedure `draw_it` takes care of drawing the image on the screen. It looks quite complex, but it's actually very simple.

```
PROCEDURE draw_it ..  
MOUSE x%,y%,k% ..
```

If the right mouse button is pressed, evaluate the position of the mouse, make sure it's within one of the grid boxes, then fill that box with the drawing color.

```
IF k%=1 ..  
IF INT(x%/10)<>x%/10 AND INT(y%/10)<>y%/10      !Check for grid ..  
tx%=(INT(x%/10))*10+1 ..  
ty%=(INT(y%/10))*10+1 ..  
DEFFILL c% ..  
PBOX tx%,ty%,tx%+8,ty%+8 ..
```

If a grid block was filled, plot the corresponding point on the actual size image which is displayed on the screen.

```
x1%=INT(x%/10) ↴  
y1%=INT(y%/10) ↴  
PLOT 250+x1%,40+y1% ↴  
GET 251,41,268,58,pic$ ↴  
pic$(pc%)=pic$ ↴  
ENDIF ↴  
ENDIF ↴  
RETURN ↴  
' ↴
```

If the user elects to save the designed object, this procedure permits the selection of saving as a .SHP or .LST file. A .SHP file may be loaded directly into a variable for use by the PUT command.

A .LST file may be merged with any GFA BASIC program, then the data may be read into a variable for use by the PUT command.

```
PROCEDURE sav_it ↴  
ALERT 2,alrt$(2),0,".SHP|.LST|Cancel",b ↴  
IF b=1 ↴  
FILESELECT "*.SHP",filename$,b$ ↴  
IF b$<>"" ↴  
OPEN "O",#1,b$ ↴  
PRINT #1,pic$ ↴  
CLOSE #1 ↴  
ENDIF ↴  
ENDIF ↴
```

Since no processing is needed to save the image as a .SHP, b\$ is simply written to the disk. Saving a .LST file is more involved, since this segment actually needs to generate GFA BASIC code which can later be merged.

```
IF b=2 ↴  
FILESELECT "*.LST",filename$,b$ ↴  
IF b$<>"" ↴
```

Change the mouse pointer to the “bee” to indicate that the processor is busy, then open the file.

```
DEFMOUSE 2
name$=MID$(b$,2,LEN(b$)-5)
ptr%=VARPTR(pic$)
OPEN "O",#1,b$
```

Now, generate the code and send it to the disk file. Information on exactly what's going on with the code generation sequence can be obtained in Chapter 2, by examining the explanations for GET and PUT.

```
PRINT #1,"Procedure "+name$ ..
PRINT #1;name$;"$ = MKI$(";+DPEEK(ptr%);")" ..
PRINT #1;name$;"$ = ";name$;"$+MKI$(";+DPEEK(ptr%+2);")" ..
PRINT #1;name$;"$ = ";name$;"$+MKI$(";DPEEK(ptr%+4);")" ..
PRINT #1,"For CTR%=0 To 288" ..
PRINT #1," Read A%" ..
PRINT #1,"   ";name$;"$=";name$;"$+Chr$(A%)" ..
PRINT #1,"Next CTR%" ..
PRINT #1,"" ..
ptr%=ptr%+6 ..
cnt%=0 ..
FOR ctr%=0 TO 272 STEP 17 ..
  d$="Data" ..
  i$="" ..
  FOR x%=ptr%+ctr% TO ptr%+16+ctr% ..
    i$=i$+STR$(PEEK(x%))+"," ..
  NEXT x% ..
  PRINT #1;d$;LEFT$(i$,LEN(i$)-1) ..
NEXT ctr% ..
PRINT #1,"Return" ..
CLOSE #1 ..
ENDIF ..
ENDIF ..
```

If the option Cancel is selected, the procedure falls through to this point, and nothing happens. As this procedure ends, the mouse pointer is restored to the arrow symbol.

```
DEFMOUSE 0 ..  
RETURN ..  
' ..
```

If the user selects the menu option to load a .SHP file, this procedure loads it into memory, using the standard file selector, then plots the shape into the grid blocks.

```
PROCEDURE load_it ..  
FILESELECT "*.SHP",filename$,b$ ..  
DEFMOUSE 2 ..  
IF b$<>"" ..  
OPEN "I",#1,b$ ..  
count%=0 ..  
DO ..  
EXIT IF EOF(#1) ..  
p%=INP(#1) ..  
pic$=pic$+CHR$(p%) ..  
LOOP ..  
CLOSE #1 ..  
@new_pic ..  
ENDIF ..  
FOR x%=252 TO 267 ..  
FOR y%=42 TO 57 ..  
c%=POINT(x%,y%) ..  
DEFFILL c% ..  
COLOR 1 ..  
FILL ((x%-252)*10)+25,((y%-42)*10+27) ..  
NEXT y% ..  
NEXT x% ..  
ENDIF ..  
DEFMOUSE 0 ..  
RETURN ..
```

Next, the picture that was loaded must be displayed, and plotted into the grid.

```
' -->
PROCEDURE new_pic -->
PUT 251,41,pic$ -->
GET 251,41,268,58,pic$(pc%) -->
FOR x%=252 TO 267 -->
  FOR y%=42 TO 57 -->
    c%=POINT(x%,y%) -->
    DEFFILL c% -->
    x1%=((x%-252)*10)+25 -->
    y1%=((y%-42)*10)+27 -->
    tx%=(INT(x1%/10))*10+1 -->
    ty%=(INT(y1%/10))*10+1 -->
    PBOX tx%,ty%,tx%+8,ty%+8 -->
  NEXT y% -->
NEXT x% -->
RETURN -->
'
```

Procedure activate_menu is very simple. It just turns on the menu bar, and defines two boxes; one permits the object to be drawn, the other box changes the pen color.

```
PROCEDURE activate_menu -->
MENU strip$() -->
ON MENU      GOSUB menu -->
ON MENU IBOX 1,22,184,163,10 GOSUB new_color -->
ON MENU IBOX 2,20,20,164,160 GOSUB draw_it -->
RETURN -->
'
```

Procedure set_colors changes the color palette using a special VDI routine. The menu is removed, since it is not active during this procedure.

```
PROCEDURE set_colors ..  
    SGET screen$ ..  
    CLS ..  
    MENU KILL ..  
    FOR ctr%=0 TO 15 ..  
        DEFFILL 1,1 ..  
        COLOR 1 ..  
        BOX 50+ctr%*10,184,60+ctr%*0,196 ..  
        DEFFILL ctr%,1 ..  
        PBOX 51+ctr%*10,185,59+ctr%*10,195 ..  
    NEXT ctr% ..  
    DEFFILL c% ..  
    PRBOX 20,20,300,50 ..  
    PRBOX 200,150,250,180 ..
```

Here's a tricky little VDI call that will be examined in more detail later.

```
@vq_color(c%,1) ..
```

Display RGB values and display screen.

```
r%=INT((r%*7)/1000) ..  
r$=HEX$(r%) ..  
g%=INT((g%*7)/1000) ..  
g$=HEX$(g%) ..  
b%=INT((b%*7)/1000) ..  
b$=HEX$(b%) ..  
DEFTEXT 1,1,,6 ..  
TEXT 10,80,"Register Number: "+STR$(c%) ..  
TEXT 10,100,"Red value: "+r$ ..  
TEXT 10,110,"Green value: "+g$ ..  
TEXT 10,120,"Blue value: "+b$ ..  
DEFTEXT c%+1 ..  
IF c%=15 ..  
    c%=0 ..  
ENDIF ..  
TEXT 210,167,"Exit" ..  
BOX 48,155,67,174 ..  
PUT 49,156,pic$ ..
```

```
DEFLINE 1,4,1,1 ..J
DEFTEXT 1,0,,4 ..J
TEXT 183,91,"Lower" ..J
TEXT 228,91,"Higher" ..J
LINE 200,97,240,97 ..J
LINE 200,107,240,107 ..J
LINE 200,117,240,117 ..J
DEFLINE 1,1,0,0 ..J
```

Track the mouse to determine if the user enters one of the defined box areas.

```
REPEAT ..J
  MOUSE x%,y%,k% ..J
  IF k%=1 ..J
    IF x%>199 AND x%<209 ..J
      IF y%>92 AND y%<101 ..J
        DEC r% ..J
        IF r%<0 ..J
          r%=7 ..J
        ENDIF ..J
      ENDIF ..J
      IF y%>103 AND y%<111 ..J
        DEC g% ..J
        IF g%<0 ..J
          g%=7 ..J
        ENDIF ..J
      ENDIF ..J
      IF y%>113 AND y%<121 ..J
        DEC b% ..J
        IF b%<0 ..J
          b%=7 ..J
        ENDIF ..J
      ENDIF ..J
    ENDIF ..J
    IF x%>231 AND x%<240 ..J
      IF y%>92 AND y%<101 ..J
        INC r% ..J
        IF r%>7 ..J
          r%=0 ..J
        ENDIF ..J
      ENDIF ..J
    ENDIF ..J
```

```
ENDIF ..  
IF y%>103 AND y%<111 ..  
INC g% ..  
IF g%>7 ..  
g%=0 ..  
ENDIF ..  
ENDIF ..  
IF y%>113 AND y%<121 ..  
INC b% ..  
IF b%>7 ..  
b%=0 ..  
  
ENDIF ..  
ENDIF ..  
ENDIF ..  
DEFTEXT 0,1,,6 ..  
TEXT 154,100,r$ ..  
TEXT 154,110,g$ ..  
TEXT 154,120,b$ ..  
r$=STR$(r%) ..  
g$=STR$(g%) ..  
b$=STR$(b%) ..  
DEFTEXT 1,1,,6 ..  
TEXT 10,100,"Red value: " +r$ ..  
TEXT 10,110,"Green value: " +g$ ..  
TEXT 10,120,"Blue value: " +b$ ..  
d%=INT((1000*r%)/7) ..  
e%=INT((1000*g%)/7) ..  
f%=INT((1000*b%)/7) ..  
@set_col_intensity(c%,d%,e%,f%) ..  
ENDIF ..  
UNTIL x%>200 AND x%<250 AND y%>150 AND y%<180 AND k%=1 ..  
SPUT screen$ ..  
@activate_menu ..  
RETURN ..  
' ..
```

Procedure `vq_color` is a VDI function call which returns the values of the current active color if function% is equal to zero. If function zero equals one, the value of the color is returned.

```

PROCEDURE vq_color(c%,function%) ..J
DPOKE CONTRL,26      !Opcode ..J
DPOKE CONTRL+2,0     !Points in ptsin array ..J
DPOKE CONTRL+4,0     !Points in ptsout array ..J
DPOKE CONTRL+6,2     !Length of intin array ..J
DPOKE CONTRL+8,0     !Length of intout array ..J
DPOKE INTIN,c%       !Color index ..J
DPOKE INTIN+2,function%   !Function 0 = set, 1 = get ..J
VDISYS ..J
r%=DPEEK(INTOUT+2) ..J
g%=DPEEK(INTOUT+4) ..J
b%=DPEEK(INTOUT+6) ..J
RETURN ..J
' ..J

```

This procedure calls the vs_color VDI function, which is used to change the color in the hardware register.

```

PROCEDURE set_col_intensity(c%,r%,g%,b%) ..J
DPOKE CONTRL,14      !Opcode ..J
DPOKE CONTRL+2,0     !Points in ptsin array ..J
DPOKE CONTRL+4,0     !Points in ptsout array ..J
DPOKE CONTRL+6,4     !Length of intin array ..J
DPOKE CONTRL+8,0     !Length of intout array ..J
DPOKE INTIN,c%       !Color index ..J
DPOKE INTIN+2,r%     !Value of red intensity (0-1000) ..J
DPOKE INTIN+4,g%     !Value of green intensity (0-1000) ..J
DPOKE INTIN+6,b%     !Value of blue intensity (0-1000) ..J
VDISYS ..J
RETURN ..J
' ..J

```

Another menu option permits the palette to be stored as either a .PLT file to be loaded into this program for further editing functions, or as a .LST file to be merged into your own programs.

```

PROCEDURE store_palette ..J
ALERT 2,alrt$(3),0,".PLT|LST",b ..J

```

If the .PLT selection was chosen, the file may later be reloaded into the Shape Editor for further modification to a predefined object, or for use when designing other objects.

```
IF b=1 ..  
FILESELECT "*.PLT","","file$ ..  
IF file$<>"" ..  
OPEN "O",#1,file$ ..
```

Just in case a mistake is made in selecting a color file later, send a code to authenticate the file.

```
PRINT #1,"0987654321"      !Code to check for legitimate file ..  
FOR ctr%=0 TO 15 ..  
    @vq_color(ctr%,1) ..  
    r$=STR$(r%) ..  
    g$=STR$(g%) ..  
    b$=STR$(b%) ..  
    PRINT #1,r$ ..  
    PRINT #1,g$ ..  
    PRINT #1,b$ ..  
NEXT ctr% ..  
CLOSE #1 ..  
ENDIF ..  
ENDIF ..
```

If the user selects the option to create a .LST file, code must be generated which can be merged with any GFA BASIC program. Be careful to type this section **EXACTLY** as listed, as it generates usable GFA BASIC CODE in an ASCII format.

```
IF b=2 ..  
FILESELECT "*.LST","","file$ ..  
IF file$<>"" ..  
OPEN "O",#1,file$ ..  
PRINT #1,"Procedure Set_colors" ..  
PRINT #1," Local CTR%" ..  
PRINT #1," For CTR%=0 to 15" ..
```

```

PRINT #1," Read R%,G%,B%" ..
PRINT #1," @set_col_intensity(CTR%,R%,G%,B%)" ..
PRINT #1," Next CTR%" ..
FOR ctr%=0 TO 15 ..
    @vq_color(ctr%,1) ..
    PRINT #1,"Data ";STR$(r%);";";STR$(g%);";STR$(b%) ..
NEXT ctr% ..
PRINT #1,"Return" ..
PRINT #1," Procedure Set_col_intensity(C%,R%,G%,B%)" ..
PRINT #1," Dpoke Contrl,14           !Opcode" ..
PRINT #1," Dpoke Contrl+2,0         !Points in ptsin array" ..
PRINT #1," Dpoke Contrl+4,0         !Points in ptsout array" ..
PRINT #1," Dpoke Contrl+6,4        !Length of intin array" ..
PRINT #1," Dpoke Contrl+8,0        !Length of intout array" ..
PRINT #1," Dpoke Intin,C%          !Color index" ..
PRINT #1," Dpoke Intin+2,R%         !Value of red intensity (0-1000)" ..
PRINT #1," Dpoke Intin+4,G%         !Value of green intensity (0-1000)" ..
PRINT #1," Dpoke Intin+6,B%         !Value of blue intensity (0-1000)" ..
PRINT #1," Vdisys" ..
PRINT #1," Return" ..
CLOSE #1 ..
ENDIF ..
ENDIF ..
RETURN ..
' ..

```

If a previously saved palette is requested, this procedure will load it and install it. Notice that a check is made for the validity code previously saved with the file.

```

PROCEDURE load_palette ..
FILESELECT "*.PLT","","file$ ..
IF file$<"" ..
    IF RIGHTS$(file$,3)<"PLT" ..
        ALERT 1,alrt$(4),1,"OK",b ..
    ELSE ..
        OPEN "I",#1,file$ ..
        INPUT #1,a$ ..
        IF a$<>"0987654321"      !Is it my code for palette file? ..
            ALERT 1,alrt$(4),1,"OK",b ..
        ENDIF ..

```

```
ctr%=0 ..  
REPEAT ..  
    INPUT #1,$ ..  
    INPUT #1,g$ ..  
    INPUT #1,b$ ..  
    r%=VAL(r$) ..  
    g%=VAL(g$) ..  
    b%=VAL(b$) ..  
    @set_col_intensity(ctr%,r%,g%,b%) ..  
    INC ctr% ..  
UNTIL ctr%=15 ..  
no_cando: ..  
CLOSE #1 ..  
ENDIF ..  
ENDIF ..  
RETURN ..
```

Now here's a really powerful feature which permits a sequence of shapes to be put in motion to see what the effect will be.

```
PROCEDURE animate ..  
LOCAL x%,ctr% ..  
SGET screen$ ..  
ALERT 1,alrt$(6),0,"2 | 3 | 4 ",b ..  
x%=b ..  
sp%=10 ..  
TEXT 200,80,"Press <Return>" ..  
TEXT 200,89,"to Stop..." ..
```

While the loop is running, check if the mouse buttons have been pressed and adjust the speed of the animation accordingly (left mouse button slows it down, right mouse button speeds it up)..

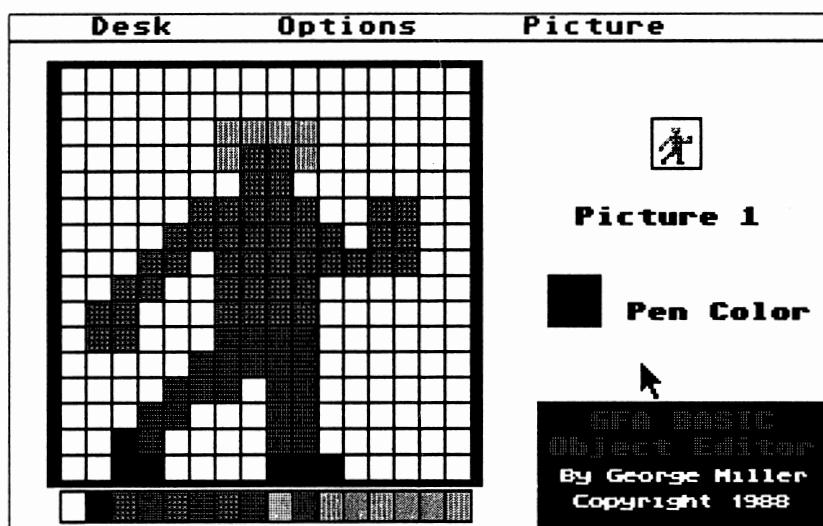
```
DO ↴  
  EXIT IF INKEY$=CHR$(13) ↴  
  
  FOR ctr%=0 TO x% ↴  
    IF MOUSEK=1 AND sp%>0           !Speed up ↴  
      DEC sp% ↴  
    ENDIF ↴  
    IF MOUSEK=2           !Slow down ↴  
      INC sp% ↴  
    ENDIF ↴  
    PAUSE sp% ↴  
    PUT 251,41,pic$(ctr%) ↴  
  NEXT ctr% ↴  
  LOOP ↴  
  SPUT screen$ ↴  
  DEFTEXT 1 ↴  
  TEXT 220,80,"Picture "+STR$(pc%+1) ↴  
RETURN ↴
```

And that's all that's needed to create this rather useful utility. An attempt was made to make it user-friendly, but some instructions may come in handy.

Using the Shape Editor

Here's some code which was generated by the shape editor to create a crude image of a man, as shown in Figure 4-1.

The name used for calling the procedure just created is the same as the name given to the shape. Notice that the code generated is not in the same format as the other programs in this book. (Keywords in all capital letters and variables in lower case letters.) This will be corrected when the code is merged into the GFA BASIC Editor/Interpreter.

Figure 4-1 MAN1.SHP

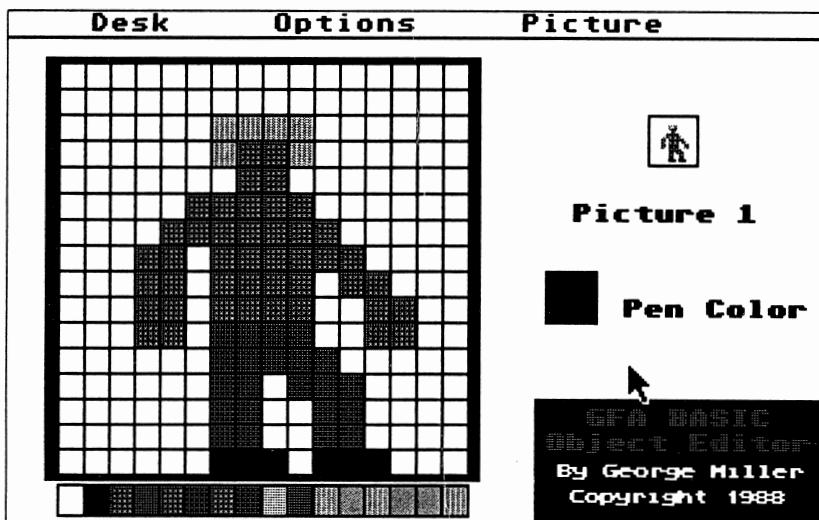
```
Procedure MAN1 ↴
MAN1$ = MKI$(17) ↴
MAN1$ = MAN1$+MKI$(17) ↴
MAN1$ = MAN1$+MKI$(4) ↴
For CTR%=0 To 288 ↴
    Read A% ↴
    MAN1$=MAN1$+Chr$(A%) ↴
Next CTR% ↴
' ↴
Data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ↴
Data 0,0,0,0,1,192,0,0,0,0,0,0,0,1,128 ↴
Data 0,0,0,0,1,192,0,0,0,0,0,0,0,1,128,0 ↴
Data 0,0,0,1,192,0,0,0,0,0,0,0,0,0,128,0,0 ↴
Data 0,0,0,0,0,0,0,0,0,0,0,0,67,224,0,0,0 ↴
Data 0,64,0,0,0,0,0,0,0,0,0,55,240,0,0,0,0 ↴
Data 0,0,0,0,0,0,0,0,0,29,216,0,0,0,0,0 ↴
Data 0,0,0,0,0,0,0,0,9,240,0,0,0,0,0,0 ↴
Data 0,0,0,0,0,0,0,0,1,224,0,0,0,0,0,0,0 ↴
Data 0,0,0,0,0,0,0,1,0,1,3,227,0,1,0,0 ↴
```

```
Data 0,0,0,0,0,0,1,0,1,7,247,0,1,0,0,0 ..J  
Data 0,0,0,0,0,1,0,1,14,61,0,1,0,0,0,0 ..J  
Data 0,0,0,0,0,0,0,7,24,0,0,0,0,0,0,0 ..J  
Data 0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0 ..J  
Data 0,0,1,128,1,128,1,128,0,0,0,0,0,0,0 ..J  
Data 0,3,0,3,0,3,0,3,0,0,0,0,0,0,0,0 ..J  
Data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ..J  
Return ..J
```

Load this shape into the Shape Editor and modify the position of it's arms and legs. This example uses only crude animation, so we won't worry too much about making it extremely smooth.

Now, after modifying the shape, save it again. The figure should resemble Figure 4-2.

Figure 4-2 MAN2.SHP



Here's the code generated by the Shape Editor:

```
Procedure MAN2 ..  
MAN2$ = MKI$(17) ..  
MAN2$ = MAN2$+MKI$(17) ..  
MAN2$ = MAN2$+MKI$(4) ..  
For CTR%=0 To 288 ..  
    Read A% ..  
    MAN2$=MAN2$+Chr$(A%) ..  
Next CTR% ..  
' ..  
Data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ..  
Data 0,0,0,0,1,192,0,0,0,0,0,0,0,0,1,128 ..  
Data 0,0,0,1,192,0,0,0,0,0,0,0,0,1,128,0 ..  
Data 0,0,1,192,0,0,0,0,0,0,0,0,0,128,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,0,1,252,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,0,1,252,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,1,236,0,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,1,236,0,0,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,1,224,0,0,0,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,0,31,224,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,0,63,224,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,48,96,0,0,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,48,96,0,0,0,0,0,0,0 ..  
Data 0,0,0,112,0,112,0,112,96,112,0,0,0,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,96,0,0,0,0,0,0,0,0,0 ..  
Data 0,0,224,0,224,0,224,0,224,0,0,0,0,0,0,0,0 ..  
Data 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 ..  
Return ..
```

Changing the Palette

To change the palette colors, select a color by pointing at the palette from the main drawing screen. Then press the left mouse button to select a new pen color. Next select "Set Colors" from the Options Menu.

The screen will change to display the set color screen.

Change the colors by pointing to the appropriate arrow heads for the R, G, and B color registers. The changes will be displayed in the color block on the screen.

When the selected color has been modified, point to the "Exit" block and press the left mouse to return to the main drawing screen.

Using The Code

The shapes created with the Shape Editor may be merged into any other program, then displayed by using the PUT command from GFA BASIC, just as in the animation sequence in the Shape Editor.

Be sure you merge each code segment created, and that your program uses PUT to display the objects on the screen.

For more information about using the PUT command, refer to Chapter 2.

A Minor Detail

This was only a brief introduction to the fascinating world of Computer Graphics. You now have the necessary tools to explore this new world, and develop truly amazing creations.

There is one more thing you need, however, for creating arcade type games, and that's a joystick routine.

Here's a simple routine that can be used in any program requiring use of joysticks.

Program 4-4 Joystick Reader

First, set up the IKBD to return the joystick packet.

```
' joystick.bas ..  
mc$=MKI$(&H23C8)+MKL$(*a%)+MKI$(&H4E75) ..  
v%=XBIOS(34)+24 ..  
o%=LPEEK(v%) ..  
LPOKE v%,VARPTR(mc$) ..  
a%=0 ..  
OUT 4,&H16 ..
```

The value of the joystick packet is stored in a% during an interrupt. The next code segment should be looped through repeatedly by your program.

```
REPEAT      !Wait for Interrupt ..  
UNTIL a% ..  
LPOKE v%,0% ..  
joy_0%=a%+1 ..  
joy_1%=a%+2 ..  
OUT 4,&H14 ..
```

Now you are all set to read the joysticks. Insert your code at this point. Remember that you must repeatedly loop through the preceding REPEAT...UNTIL loop to read the joystick.

The rest of the demonstration program is just to help you learn how the values are used.

```
PRINT AT(1,20);"Press any key to quit"; ..
REPEAT ..
    PRINT AT(1,9);"Last: Joystick ";(PEEK(a%)+1) ..
        @output(PEEK(joy_1%)) ..
    UNTIL INKEY$<>"" ..
    OUT 4,8 ..
    PROCEDURE output(x%) ..
        IF x% AND 128 ..
            PRINT "Button "; ..
        ENDIF ..
        IF x% AND 1 ..
            PRINT "Up "; ..
        ENDIF ..
        IF x% AND 2 ..
            PRINT "Down "; ..
        ENDIF ..
        IF x% AND 4 ..
            PRINT "Left "; ..
        ENDIF ..
        IF x% AND 8 ..
            PRINT "Right "; ..
        ENDIF ..
        PRINT CHR$(27);"K" ..
    RETURN ..
```

Again, the routine is actually a lot simpler than it looks. Just plug this code into your own programs, and create an original arcade action game. This routine is similar to the ones used in many arcade games.

Another excellent tool for graphics programming is PIC_CLIP.PRG, by Jim Luczak. This program allows portions of Degas™ and NEOchrome™ pictures to be "clipped" and merged with your programs, in the same manner as the objects created with the Shape Editor, only the objects can be as large as the entire screen. This program, and many others, may be obtained from many Public Domain software sources, such as the MICHTRON Round Table on GEnie™.

Now you have all the tools and are ready to create stunning visual effects, but there's still one important facet missing from your game. Sound, and that's our next stop in the world of GFA BASIC.

Chapter 5

Sound



Chapter 5

Sound

Walk through any shopping mall or airport, and it won't be long before you hear the electronic sounds of the arcades. No matter how impressive the graphics, a game is not finished until it has a voice.

Although the ST's sound chip is rather old in terms of the latest technology; it is still a powerful tool once you learn how to use it.

The sound chip can play three voices, or sounds at once. It's also possible to send any of 4096 different sounds through any one of the three voices, and adjust the tone of each sound by changing its wave type.

The Physics of Sound

Before we discuss ways to use the sound commands of GFA BASIC, let's discuss what a sound is, and how the ST's sound chip actually works.

Any sound you can hear, whether it's from a natural source or electronically generated source is just a vibration of the air. The inner ear detects these vibrations, and passes them to the brain as a sensation of sound.

Sound may be defined as a displacement of pressure in the atmosphere. This displacement may be graphed as a function of the changes in pressure during a period of time. By labeling one axis as time, and the other as pressure displacement, a simple graphic representation of a sound, called a waveform, can be drawn.

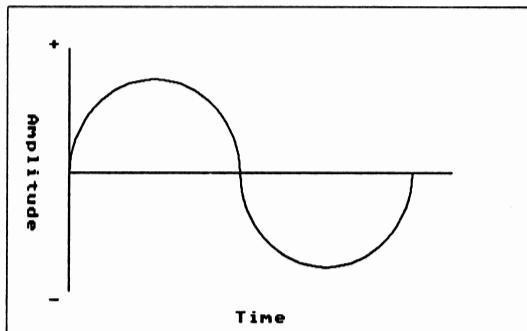
The Sine Wave

A sine wave, as displayed in Figure 5-1, raises smoothly to a maximum, or peak, value, then falls to its minimum value. One time through the process (rising, falling, and rising again to its starting value) is called a cycle.

The rate at which these cycles occur is called a frequency. Frequency may be expressed in cycles per second, or more commonly, as Hertz (Hz).

A sine wave is considered an analog sound (varying levels of amplitude), and can only be imitated by the digital computer.

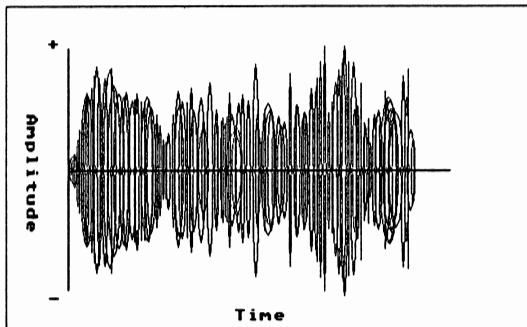
Figure 5-1 Sine Wave



Understanding Waveforms

When a note is played, it consists of a sine wave vibrating at a fundamental frequency, and the harmonics of that frequency. Figure 5-2 represents an actual sound wave, although it is somewhat simplified here.

Figure 5-2 Sound Wave



The fundamental frequency defines the overall pitch of the note, while harmonics are sine waves having frequencies which are integer multiples of the fundamental frequency.

Let's say the fundamental frequency is 1000 Hertz. The second harmonic frequency is twice the fundamental frequency, or 2000 Hertz. The third harmonic is three times the fundamental frequency, or 3000 Hertz. And so on for each harmonic value.

Acoustic instruments, like the violin or guitar, have complicated harmonic patterns, with many harmonics.

The Sound Command

The SOUND command is explained in Chapter 2. The SOUND command consists of five parts.

SOUND *voice, volume, note, octave, duration*

- voice* A numeric expression representing the sound channel to be used. The only valid parameters are 1, 2, or 3.
- volume* A numeric expression between 0 and 15 representing the volume at which the sound is to be played. When the volume is 0, the sound is off, 15 is the highest volume.
- note* This is a numeric expression between 1 and 12 representing the note to be played.

<u>Number</u>	<u>Note</u>
1	C
2	C#
3	D
4	D#
5	E
6	F
7	F#
8	G
9	G#
10	A
11	A#
12	B

octave This is a numeric expression between 1 and 8 which represents the value of the octave to be played. 1 represents the lowest octave, 8 is the highest.

duration This is an integer expression and specifies the time in 50'ths of a second to wait before executing the next command in sequence.

Here's a simple demonstration program using just the SOUND command.

Program 5-1 Simple Sounds

```
FOR ctr%=1 TO 12 ..  
    SOUND 1,15,ctr%,4,50 ..  
NEXT ctr% ..  
SOUND 1,0 ..
```

This simple routine plays each note in the fourth octave, and pauses for one second before continuing. After all twelve notes are played, the volume of voice 1 is set to zero.

If the command SOUND 1,0 where not included, the sound chip would continue to generate a sound until a key was pressed

Program 5-2 A Simple Tune

A better use of the sound command is to play a simple little tune. as in this program.

```
RESTORE song ..  
DO ..  
    READ n%,o% ..  
    EXIT IF n%=0 ..  
    SOUND 1,15,n%,o%,5 ..  
LOOP ..  
SOUND 1,15,1,6,18 ..  
SOUND 1,0 ..  
' ..  
song: ..  
DATA 8,4,1,4,10,4,8,4,1,4,1,5 ..  
DATA 8,4,1,4,10,4,8,4,1,4,1,5 ..  
DATA 8,4,1,4,10,4,8,4,1,4,5,5,1,5 ..  
DATA 3,5,5,5,6,5,8,5,10,5,12,5,0,0 ..
```

This example reads in a series of notes and octaves from the data statements and plays them.

The SOUND command can also be used to produce sound effects, as demonstrated by Program 5-3.

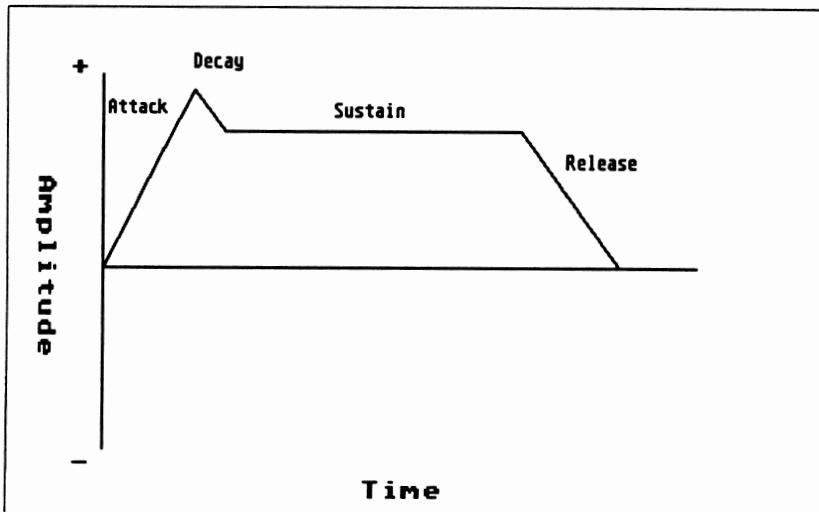
Program 5-3 Sound Effect

```
start: l
FOR ctr% = 1 TO 12 l
  SOUND 1,15,ctr%,4,0 l
NEXT ctr% l
FOR ctr% = 11 TO 2 STEP -1 l
  SOUND 1,15,ctr%,540 l
NEXT ctr% l
GOTO start l
```

The Sound Envelope

Every sound is made up of four parts. First is the attack, which occurs from the beginning of the sound until it reaches its maximum value. Then the sound begins to fall from the peak to some mid volume range. The rate at which this fall occurs is called decay. The time period when the sound is maintained at this mid volume level is called the sustain level. Finally, the period of time between when the sound is no longer generated, and the volume falls to the zero level is called the release. Figure 5-3 illustrates these sound parameters.

Figure 5-3 The ADSR Envelope



WAVE

The WAVE function is used to provide access to multivoice music and sound capabilities of GFA BASIC. WAVE does not produce any sound, but defines the sound which will be produced by using the SOUND function.

WAVE voice,env,form,len,dur

voice An integer expression between zero and 32 which determines which combination of the three sound channels will be turned on, by setting specific bits. 256 times the period of the noise generator (zero to 31) may also be added to the value of the voice.

<u>Decimal</u>	<u>Binary</u>	<u>Channel</u>
0	00000000	All channels OFF
1	00000001	Channel 1 ON
2	00000010	Channel 2 ON
4	00000100	Channel 3 ON
8	00001000	Channel 1 NOISE ON
16	00010000	Channel 2 NOISE ON
32	00100000	Channel 3NOISE ON

env An integer expression which defines the channels for which the envelope is active. This is also a bitwise value.

<u>Decimal</u>	<u>Binary</u>	<u>Channel</u>
1	00000001	Channel 1 Envelope ON
2	00000010	Channel 2 Envelope ON
4	00000100	Channel 3 Envelope ON

<i>form</i>	An integer expression which specifies the shape of the wave.																						
	<table><thead><tr><th><u>Value</u></th><th><u>Description</u></th></tr></thead><tbody><tr><td>0-3</td><td>Linear, falling</td></tr><tr><td>4-7</td><td>Linear, rising, terminating</td></tr><tr><td>8</td><td>Sawtooth, rising</td></tr><tr><td>9</td><td>Linear, falling</td></tr><tr><td>10</td><td>Peaked, begin falling</td></tr><tr><td>11</td><td>Linear, falling, variable volume</td></tr><tr><td>12</td><td>Sawtooth, rising</td></tr><tr><td>13</td><td>Linear, rising, continuous</td></tr><tr><td>14</td><td>Peaked, begin rising</td></tr><tr><td>15</td><td>Linear rising</td></tr></tbody></table>	<u>Value</u>	<u>Description</u>	0-3	Linear, falling	4-7	Linear, rising, terminating	8	Sawtooth, rising	9	Linear, falling	10	Peaked, begin falling	11	Linear, falling, variable volume	12	Sawtooth, rising	13	Linear, rising, continuous	14	Peaked, begin rising	15	Linear rising
<u>Value</u>	<u>Description</u>																						
0-3	Linear, falling																						
4-7	Linear, rising, terminating																						
8	Sawtooth, rising																						
9	Linear, falling																						
10	Peaked, begin falling																						
11	Linear, falling, variable volume																						
12	Sawtooth, rising																						
13	Linear, rising, continuous																						
14	Peaked, begin rising																						
15	Linear rising																						
<i>len</i>	An integer expression which specifies the length of time during which the envelope is to be played.																						
<i>dur</i>	An integer expression which specifies the time, in 50ths of a second, to delay before executing the next command																						

Any parameter which is to remain unchanged can be omitted. The statement WAVE 0,0 may be used to turn off all sound channels.

The best way to gain familiarity with using sound effects is through experimentation. This next program also demonstrates one method for constructing a friendly user interface.

Program 5-4 Sound Edit

Start off by reserving an area of memory for the necessary overhead, variable storage, and alert boxes. Then, as usual, save the default desktop palette so it may be restored when the program ends.

```
DIM palette%(15),vol%(3),n%(3),oct%(3),dur%(3) ↴
RESERVE FRE(0)-32000    !Free space reserved for Alerts ↴
@save_palette ↴
CLS ↴
' ↴
' Check for proper (Medium) Resolution ↴
' ↴
```

Due to the graphics used in designing the screen, Sound Edit will only run in medium resolution. Therefore, check for medium resolution. If not found, then issue a warning in the form of an alert box, and arrange an orderly exit.

```
rez%=XBIOS(4) ↴
IF rez%<>1 ↴
  alrt$="Sound Editor only|runs in Medium Rez." ↴
  ALERT 1,alrt$,1,"OK",b ↴
  EDIT ↴
  END ↴
ENDIF ↴
' ↴
```

If everything is okay so far, draw the program screen. This is done by a subroutine. Since this routine is only called once, it could have been placed here, and a subroutine call avoided. However, as a matter of personal preference, it has been placed near the end of the program. The draw_screen procedure will be explained in more detail when it is encountered in the actual code.

```
@draw_screen ↴
' ↴
```

Here's the main loop of Sound Edit. This is the driver routine which monitors the mouse and calls all of the needed procedures.

When designing and testing a graphics screen which requires input from the mouse, it's often handy to install a means of testing the mouse position in the program. Later this piece of code would have been deleted. For demonstration purposes, the test and mouse interface debugging code was left intact, although it is nonfunctional due to the REM statements. This may give you some ideas of how to test and debug your own works in progress.

```
DO ↴  
  MOUSE x%,y%,k% ↴  
  IF k% AND 1 ↴  
    @check_mouse ↴  
    REPEAT ↴  
      PAUSE 10 ↴  
    UNTIL NOT MOUSEK ↴  
    ↴  
    REM This is not needed. It's a demo segment which ↴  
    REM prints the mouse position on the screen whenever ↴  
    REM the left mouse button is pressed. ↴  
    REM This is handy whenever designing a graphic screen, or ↴  
    REM when you need to check for graphic coordinates. ↴  
    ↴  
    REM PRINT AT(65,1);SPACE$(14) ↴  
    REM PRINT AT(65,1);"x=";x%; " y=";y% ↴  
    ↴  
  ENDIF ↴
```

The PAUSE 10 statement in the preceding segment will allow enough time for most users to release the mouse button before the program loop returns to this point to prevent multiple, accidental button presses. Since the sounds created may not be exactly harmonious, and at times you may wish to turn the sound off before the entire sound has played, code has been included to permit the sound to be turned off. Just press the right mouse button. WAVE 0,0 turns off all voices.

```
  IF k% AND 2 ↴  
    WAVE 0,0 ↴  
  ENDIF ↴  
LOOP ↴
```

The first procedure in the program checks the coordinates of the mouse after the left mouse button has been pressed. ($k\% = 1$) The y position of the mouse ($y\%$) is checked first for the vertical alignment of the mouse. If any of the defined areas on the screen are selected, the appropriate procedure is called, where further processing of the x coordinate is conducted. First the left side of the screen is checked for selection of the elements of the WAVE command.

```
'  
PROCEDURE check_mouse ..  
'check WAVE ..  
IF y%>31 AND y%<42 ..  
    @voice ..  
ENDIF ..  
IF y%>52 AND y%<62 ..  
    @noise ..  
ENDIF ..  
IF y%>72 AND y%<82 ..  
    @envelope ..  
ENDIF ..  
IF y%>90 AND y%<103 ..  
    @length ..  
ENDIF ..  
IF y%>110 AND y%<123 ..  
    @duration ..  
ENDIF ..  
IF y%>150 AND y%<160 ..  
    @wave_1 ..  
ENDIF ..  
IF y%>165 AND y%<175 ..  
    @wave_2 ..  
ENDIF ..  
'..
```

Further checks are made for possible matches of the defined blocks on the right (SOUND) side of the screen. On the right side of the screen, the parameters for the different available notes may be selected.

```
' Was a note selected? ..  
IF y%>110 AND y%<121 AND x%>331 ..  
    @note_1_select ..  
ENDIF ..  
IF y%>126 AND y%<137 AND x%>331 ..  
    @note_2_select ..  
ENDIF ..  
' ..  
' Was Duration selected? ..  
IF y%>141 AND y%<153 ..  
    @sound_duration ..  
ENDIF ..
```

After all checks and subsequent procedures have been called, the results are displayed in the command boxes on the screen.

```
', ..  
' Display results ..  
' WAVE settings ..  
v$=STR$(voc%) ..  
e$=STR$(env%) ..  
w$=STR$(wave%) ..  
l$=STR$(length%) ..  
d$=STR$(dur%) ..  
@wave_string ..  
, ..  
@sound_string ..  
,
```

If the box at the top center of the screen, marked "Play" is selected, the values defined are passed to the SOUND and WAVE commands, and a sound sample is played.

```
IF y%>13 AND y%<25 AND x%>299 AND x%<376 ..  
    SOUND 1,vol%(1),n%(1),oct%(1),dur%(1) ..  
    SOUND 2,vol%(2),n%(2),oct%(2),dur%(2) ..  
    SOUND 3,vol%(3),n%(3),oct%(3),dur%(3) ..  
    IF wave%<>0 ..  
        WAVE voc%,env%,wave%,length%,dur% ..  
    ENDIF ..  
ENDIF ..  
' ..
```

If the Exit box was selected, check to ascertain that the button was not pressed in error, then do clean up by clearing the screen, restoring the original palette, turn off any sound channels, release the reserve 32K segment of memory, and exit the program.

```
IF y%>52 AND y%<103 AND x%>517 AND x%<637 ..  
    ALERT 1,"Really Quit?",2,"Yes|No",b ..  
    IF b=1 ..  
        CLS ..  
        @restore_palette ..  
        WAVE 0,0    !Turn off the sound before leaving ..  
        RESERVE FRE(0)+32000 ..  
    END ..  
ENDIF ..  
ENDIF ..
```

Check for the right mouse button again, just to get a little faster response on killing the sound.

```
IF MOUSEK=2 ..  
    WAVE 0,0 ..  
ENDIF ..  
RETURN ..  
' ..
```

If the y coordinates for the voice area matched, check for the proper x coordinates. If the x coordinates match, change the button color from blue to red, and set the bitwise value for the selected voice.

```
PROCEDURE voice ..  
  ' Is it WAVE? ..  
  IF x%>120 AND x%<154 ..  
    ctr%=0 ..  
    IF POINT(121,39)=2 ..  
      DEFFILL 3 ..  
    ELSE ..  
      DEFFILL 2 ..  
    ENDIF ..  
    PBOX 120+ctr%*50,32,154+ctr%*50,42 ..  
    voc%=voc% XOR &X1 ..  
  ENDIF ..  
  ..  
  IF x%>167 AND x%<204 ..  
    ctr%=1 ..  
    IF POINT(175,39)=2 ..  
      DEFFILL 3 ..  
    ELSE ..  
      DEFFILL 2 ..  
    ENDIF ..  
    PBOX 120+ctr%*50,32,154+ctr%*50,42 ..  
    voc%=voc% XOR &X10 ..  
  ENDIF ..  
  ..  
  IF x%>219 AND x%<253 ..  
    ctr%=2 ..  
    IF POINT(221,39)=2 ..  
      DEFFILL 3 ..  
    ELSE ..  
      DEFFILL 2 ..  
    ENDIF ..  
    PBOX 120+ctr%*50,32,154+ctr%*50,42 ..  
    voc%=voc% XOR &X100 ..  
  ENDIF ..
```

Since drawing the box red will overwrite the numerals, display them again to refresh the screen.

```
TEXT 128,40," 1  2  3" ..
```

Since the y coordinates will also match up for the voice selections of the SOUND command, check the x coordinates for the sound voices. Since there is a possibility that a sound channel was already defined, and we'll want to display the data properly, the pointer to the value in the array is stored as opt% (old pointer). Later, a check will be made as to whether a refresh of the screen is needed.

```
' Is it SOUND? ..  
' ..  
opt% = pt% ..
```

First, Voice 1:

```
IF x% > 411 AND x% < 447 ..  
PUT 330,31,s$ !Clear Voice Selector ..  
ctr% = 0 ..  
DEFFILL 2 ..  
PBOX 412 + ctr% * 50,32,446 + ctr% * 50,42 !Paint Red for 'selected' ..  
pt% = 1 ..  
ENDIF ..
```

Voice 2:

```
IF x% > 461 AND x% < 498 ..  
PUT 330,31,s$ ..  
ctr% = 1 ..  
DEFFILL 2 ..  
PBOX 462,32,496,42 ..  
pt% = 2 ..  
ENDIF ..
```

Voice 3:

```
IF x%>510 AND x%<547 ..  
PUT 330,31,$$ ..  
ctr%=2 ..  
DEFFILL 2 ..  
PBOX 511,32,546,42 ..  
pt%=3 ..  
ENDIF ..  
' ..  
TEXT 419,40," 1 2 3" ..
```

If a new voice was selected, the pointer (pt%) will have changed, and a screen refresh will be needed to show the proper values for the new voice.

```
IF opt%<>pt% ..  
@show_note ..  
ENDIF ..  
RETURN ..  
' ..
```

Check for the x coordinates of the Noise selection and carry out the appropriate changes as before. Again, this is a bitwise operation, so an XOR (exclusive OR) is used to set or clear the bits.

```
PROCEDURE noise ..  
' Was NOISE selected? ..  
IF x%>120 AND x%<154 ..  
ctr%=0 ..  
IF POINT(121,59)=2 ..  
DEFFILL 3 ..  
ELSE ..  
DEFFILL 2 ..  
ENDIF ..  
PBOX 120+ctr%*50,52,154+ctr%*50,62 ..  
voc%=voc% XOR &X1000 ..  
ENDIF ..  
' ..  
IF x%>167 AND x%<204 ..  
ctr%=1 ..
```

```
IF POINT(175,59)=2 ..  
    DEFFILL 3 ..  
ELSE ..  
    DEFFILL 2 ..  
ENDIF ..  
PBOX 120+ctr%*50,52,154+ctr%*50,62 ..  
voc%=voc% XOR &X10000 ..  
ENDIF ..  
' ..  
IF x%>219 AND x%<253 ..  
    ctr%=2 ..  
    IF POINT(221,59)=2 ..  
        DEFFILL 3 ..  
    ELSE ..  
        DEFFILL 2 ..  
    ENDIF ..  
    PBOX 120+ctr%*50,52,154+ctr%*50,62 ..  
    voc%=voc% XOR &X10000 ..  
ENDIF ..  
TEXT 128,60," 1 2 3" ..
```

The volume buttons for the SOUND command are also located on these y coordinates, so a check is made for them, too.

```
' ..  
' Was Volume selected? ..  
IF x%>412 AND x%<447 ..  
    INC vol%(pt%) ..
```

Only volume levels of 0 through 15 are valid. A check is necessary to make sure the proper levels aren't exceeded. This routine sets up a "roll over" to 0 or 15 if the changes exceed prescribed values.

```
IF vol%(pt%)>15 ..  
    vol%(pt%)=0 ..  
ENDIF ..  
ENDIF ..  
IF x%>462 AND x%<498 ..
```

```
DEC vol%(pt%) ..  
IF vol%(pt%)<0 ..  
    vol%(pt%)=15 ..  
ENDIF ..  
ENDIF ..  
RETURN ..  
' ..
```

Check for possible changes to the WAVE envelope parameters.

```
PROCEDURE envelope ..  
' Was Envelope selected? ..  
IF x%>120 AND x%<154 ..  
    ctr%=0 ..  
    IF POINT(121,79)=2 ..  
        DEFFILL 3 ..  
    ELSE ..  
        DEFFILL 2 ..  
    ENDIF ..  
    PBOX 120+ctr%*50,72,154+ctr%*50,82 ..  
    env%=env% XOR &X1 ..  
ENDIF ..  
' ..  
IF x%>167 AND x%<204 ..  
    ctr%=1 ..  
    IF POINT(175,79)=2 ..  
        DEFFILL 3 ..  
    ELSE ..  
        DEFFILL 2 ..  
    ENDIF ..  
    PBOX 120+ctr%*50,72,154+ctr%*50,82 ..  
    env%=env% XOR &X10 ..  
ENDIF ..  
' ..  
IF x%>219 AND x%<253 ..  
    ctr%=2 ..  
    IF POINT(221,79)=2 ..  
        DEFFILL 3 ..  
    ELSE ..  
        DEFFILL 2 ..
```

```
ENDIF ↴
PBOX 120+ctr%*50,72,154+ctr%*50,82 ↴
env%=env% XOR &X100 ↴
ENDIF ↴
TEXT 128,80," 1 2 3" ↴
' ↴
```

Again, for simplicity, the octave value of the Sound command is on these same y-coordinates. Check for changes in the octave values now. Notice that again a "roll over" routine was used.

```
' Was Octave selected? ↴
IF x%>412 AND x%<447 ↴
  INC oct%(pt%) ↴
  IF oct%(pt%)>8 ↴
    oct%(pt%)=1 ↴
  ENDIF ↴
  IF oct%(pt%)<1 ↴
    oct%(pt%)=8 ↴
  ENDIF ↴
ENDIF ↴
IF x%>462 AND x%<498 ↴
  DEC oct%(pt%) ↴
  IF oct%(pt%)>8 ↴
    oct%(pt%)=1 ↴
  ENDIF ↴
  IF oct%(pt%)<1 ↴
    oct%(pt%)=8 ↴
  ENDIF ↴
ENDIF ↴
RETURN ↴
' ↴
```

Now, check duration:

```
PROCEDURE duration ↴
  ' Was Duration selected? ↴
  IF x%>119 AND x%<154 ↴
    INC dur% ↴
    IF dur%>65535 ↴
      dur%=0 ↴
    ENDIF ↴
  ENDIF ↴
  IF x%>169 AND x%<204 ↴
    DEC dur% ↴
    IF dur%<0 ↴
      dur%=65535 ↴
    ENDIF ↴
  ENDIF ↴
  RETURN ↴
' ↴
```

Length is another parameter of the WAVE command. This procedure checks for any selected changes in the length value.

```
PROCEDURE length ↴
  IF x%>119 AND x%<154 ↴
    INC length% ↴
    IF length%>65535 ↴
      length%=0 ↴
    ENDIF ↴
  ENDIF ↴
  IF x%>169 AND x%<204 ↴
    DEC length% ↴
    IF length%<0 ↴
      length%=65535 ↴
    ENDIF ↴
  ENDIF ↴
  RETURN ↴
' ↴
```

Ten different wave forms may be selected for the WAVE command. More precise information about these wave forms maybe found in Chapter 2. Two sets of y-coordinates correspond to the screen rows of the displayed selectable waveforms. The next two procedures take care of any necessary processing.

```
PROCEDURE wave_1 ↴
  @wave_boxes ↴
  IF x%>35 AND x%<75 ↴
    wave%=3 ↴
    PUT 34,149,r$ ↴
  ENDIF ↴
  IF x%>85 AND x%<125 ↴
    wave%=4 ↴
    PUT 84,149,r$ ↴
  ENDIF ↴
  IF x%>135 AND x%<175 ↴
    wave%=8 ↴
    PUT 134,149,r$ ↴
  ENDIF ↴
  IF x%>185 AND x%<225 ↴
    wave%=9 ↴
    PUT 184,149,r$ ↴
  ENDIF ↴
  IF x%>235 AND x%<275 ↴
    wave%=10 ↴
    PUT 234,149,r$ ↴
  ENDIF ↴
  @waveform ↴
RETURN ↴
' ↴
PROCEDURE wave_2 ↴
  @wave_boxes ↴
  IF x%>35 AND x%<75 ↴
    wave%=11 ↴
    PUT 34,164,r$ ↴
  ENDIF ↴
  IF x%>85 AND x%<125 ↴
    wave%=12 ↴
    PUT 84,164,r$ ↴
  ENDIF ↴
  IF x%>135 AND x%<175 ↴
```

```
wave%=13 ..  
PUT 134,164,r$ ..  
ENDIF ..  
IF x%>185 AND x%<225 ..  
wave%=14 ..  
PUT 184,164,r$ ..  
ENDIF ..  
IF x%>235 AND x%<275 ..  
wave%=15 ..  
PUT 234,164,r$ ..  
ENDIF ..  
@waveform ..  
RETURN ..  
' ..
```

The next two procedures are called for processing changes in the note values of the SOUND command. Again the graphics coordinates are checked for the x position of the mouse.

```
PROCEDURE note_1_select ..  
@note_box ..  
IF x%>331 AND x%<372 ..  
n%(pt%)=1 ..  
PUT 330,110,r$ ..  
ENDIF ..  
IF x%>381 AND x%<421 ..  
n%(pt%)=2 ..  
PUT 380,110,r$ ..  
ENDIF ..  
IF x%>431 AND x%<471 ..  
n%(pt%)=3 ..  
PUT 430,110,r$ ..  
ENDIF ..  
IF x%>481 AND x%<521 ..  
n%(pt%)=4 ..  
PUT 480,110,r$ ..  
ENDIF ..  
IF x%>531 AND x%<571 ..  
n%(pt%)=5 ..  
PUT 530,110,r$ ..
```

```
ENDIF ↴
IF x%>581 AND x%<622 ↴
  n%(pl%)=6 ↴
  PUT 580,110,r$ ↴
ENDIF ↴
@notes ↴
RETURN ↴
' ↴
PROCEDURE note_2_select ↴
@note_box ↴
IF x%>331 AND x%<372 ↴
  n%(pl%)=7 ↴
  PUT 330,126,r$ ↴
ENDIF ↴
IF x%>381 AND x%<421 ↴
  n%(pl%)=8 ↴
  PUT 380,126,r$ ↴
ENDIF ↴
IF x%>431 AND x%<471 ↴
  n%(pl%)=9 ↴
  PUT 430,126,r$ ↴
ENDIF ↴
IF x%>481 AND x%<521 ↴
  n%(pl%)=10 ↴
  PUT 480,126,r$ ↴
ENDIF ↴
IF x%>531 AND x%<571 ↴
  n%(pl%)=11 ↴
  PUT 530,126,r$ ↴
ENDIF ↴
IF x%>581 AND x%<622 ↴
  n%(pl%)=12 ↴
  PUT 580,126,r$ ↴
ENDIF ↴
@notes ↴
RETURN ↴
' ↴
```

Sometimes values for a previously defined note need to be modified. Display the correct, present note setting before any changes are made. The variable r\$ is a previously stored red box. (Saved by using the GET function during the initialization process.)

```
PROCEDURE show_note ↴
  ' Shows current note selected for voice ↴
  @note_box ↴
  IF n%(pl%)=1 ↴
    PUT 330,110,r$ ↴
  ENDIF ↴
  IF n%(pl%)=2 ↴
    PUT 380,110,r$ ↴
  ENDIF ↴
  IF n%(pl%)=3 ↴
    PUT 430,110,r$ ↴
  ENDIF ↴
  IF n%(pl%)=4 ↴
    PUT 480,110,r$ ↴
  ENDIF ↴
  IF n%(pl%)=5 ↴
    PUT 530,110,r$ ↴
  ENDIF ↴
  IF n%(pl%)=6 ↴
    PUT 580,110,r$ ↴
  ENDIF ↴
  IF n%(pl%)=7 ↴
    PUT 330,126,r$ ↴
  ENDIF ↴
  IF n%(pl%)=8 ↴
    PUT 380,126,r$ ↴
  ENDIF ↴
  IF n%(pl%)=9 ↴
    PUT 430,126,r$ ↴
  ENDIF ↴
  IF n%(pl%)=10 ↴
    PUT 480,126,r$ ↴
  ENDIF ↴
  IF n%(pl%)=11 ↴
    PUT 530,126,r$ ↴
  ENDIF ↴
```

```
IF n%(pt%)=12 ..  
    PUT 580,126,r$ ..  
ENDIF ..  
@notes ..  
RETURN ..  
' ..
```

If either of the boxes for Sound duration was selected, the values are processed in the next procedure. Once again, a "roll over" is utilized to keep within the proper limits.

```
PROCEDURE sound_duration ..  
IF x%>412 AND x%<448 ..  
    INC dur%(pt%) ..  
    IF dur%(pt%)>65535 ..  
        dur%(pt%)=0 ..  
    ENDIF ..  
ENDIF ..  
IF x%>462 AND y%<499 ..  
    DEC dur%(pt%) ..  
    IF dur%(pt%)<0 ..  
        dur%(pt%)=65535 ..  
    ENDIF ..  
ENDIF ..  
RETURN ..  
' ..
```

This is the actual initialization process. The screen is drawn, colors are defined, and preliminary values are assigned.

```
PROCEDURE draw_screen ..  
' ..  
SETCOLOR 0,0,0 ..  
SETCOLOR 1,7,0,0 ..  
SETCOLOR 2,0,0,7 ..  
SETCOLOR 3,7,7,7 ..  
GRAPHMODE 2 ..  
DEFTEXT 2,17,,32 ..  
TEXT 50,24,"Sound Editor" ..
```

An interesting touch. Since Outlined characters were used in the title, the blank space inside the characters is filled with color 3. The coordinates of the fill locations are stored in data statements and read by the FOR...NEXT loop.

```
DEFFILL 3 ..  
FOR ctr% = 0 TO 11 ..  
    READ x%,y% ..  
    FILL x%,y% ..  
NEXT ctr% ..  
COLOR 2 ..  
PBOX 300,14,375,24 ..  
BOX 299,13,376,25 ..  
COLOR 1 ..  
' ..  
DATA 61,15,79,21,92,21,108,20,133,20,163,11,181,9 ..  
DATA 203,2,201,8,218,7,238,8,254,9 ..  
' ..  
DEFTEXT 1,1,,6 ..  
TEXT 320,21,"Play" ..  
TEXT 350,6,"GFA BASIC Programmers Reference Guide" ..  
TEXT 472,15,"Vol. I" ..  
TEXT 430,24,"By George Miller" ..  
DRAW 1,27 TO 639,27 ..  
DRAW 320,27 TO 320,199 ..  
DRAW 321,27 TO 321,199 ..  
' ..  
' Voice selector for WAVE ..  
' ..  
DEFFILL 2 ..  
PBOX 39,32,95,42 ..  
BOX 38,31,96,43 ..  
TEXT 40,40,"Voice" ..  
DEFFILL 3 ..  
FOR ctr% = 0 TO 2 ..  
    BOX 119+ctr%*50,31,155+ctr%*50,43 ..  
    PBOX 120+ctr%*50,32,154+ctr%*50,42 ..  
NEXT ctr% ..  
TEXT 128,40," 1 2 3" ..  
' ..
```

' Voice Selector for SOUND ..
' ..

Take a short cut here. Copy the box just drawn and save it to use later.

' Short, copy other Voice selector ..
GET 38,31,319,44,s\$..
PUT 330,31,s\$..
' ..
DEFFILL 2 ..
PBOX 39,52,95,62 ..
BOX 38,51,96,63 ..
TEXT 40,60," Noise" ..
DEFFILL 3 ..
FOR ctr%=0 TO 2 ..
 BOX 119+ctr%*50,51,155+ctr%*50,63 ..
 PBOX 120+ctr%*50,52,154+ctr%*50,62 ..
NEXT ctr% ..
TEXT 128,60," 1 2 3" ..
' ..
DEFFILL 2 ..
PBOX 39,72,95,82 ..
BOX 38,71,96,83 ..
TEXT 40,80," Env" ..
DEFFILL 3 ..
FOR ctr%=0 TO 2 ..
 BOX 119+ctr%*50,71,155+ctr%*50,83 ..
 PBOX 120+ctr%*50,72,154+ctr%*50,82 ..
NEXT ctr% ..
TEXT 128,80," 1 2 3" ..
' ..
DEFFILL 2 ..
PBOX 39,92,95,102 ..
BOX 38,91,96,103 ..
DEFFILL 3 ..
FOR ctr%=0 TO 1 ..
 BOX 119+ctr%*50,91,155+ctr%*50,103 ..
 PBOX 120+ctr%*50,92,154+ctr%*50,102 ..
NEXT ctr% ..
TEXT 133,100,CHR\$(1) ..

```
TEXT 183,100,CHR$(2) ..  
DEFFILL 1 ..  
FILL 136,97 ..  
FILL 187,98 ..  
' ..
```

Grab a copy of this box to use later, too.

```
' Box needed for SOUND command functions ..  
' Get copy ..  
' ..  
GET 37,90,300,104,1$ ..  
PUT 330,50,1$ ..  
PUT 330,70,1$ ..  
PUT 330,140,1$ ..  
' ..  
TEXT 42,100,"Length"    !for WAVE ..  
TEXT 335,80,"Octave"    !for SOUND ..  
TEXT 335,60,"Volume"    !for SOUND ..  
TEXT 350,150,"Dur" ..  
' ..  
DEFFILL 2 ..  
PBOX 39,112,95,122 ..  
BOX 38,111,96,123 ..  
TEXT 52,120,"Dur" ..  
DEFFILL 3 ..  
FOR ctr%=0 TO 1 ..  
    BOX 119+ctr%*50,111,155+ctr%*50,123 ..  
    PBOX 120+ctr%*50,112,154+ctr%*50,122 ..  
NEXT ctr% ..  
TEXT 133,120,CHR$(1) ..  
TEXT 183,120,CHR$(2) ..  
DEFFILL 1 ..  
FILL 136,117 ..  
FILL 187,118 ..  
' ..  
DEFFILL 2 ..  
PBOX 125,132,200,142 ..  
BOX 124,131,201,143 ..
```

```
TEXT 148,140,"Form" ↴
' ↴
BOX 35,150,75,160 ↴
DEFFILL 2 ↴
PBOX 36,151,74,159 ↴
GET 34,149,76,161,r$ ↴
DEFFILL 3 ↴
PBOX 36,151,74,159 ↴
GET 34,149,76,161,g$ ↴
' ↴
@wave_boxes ↴
@waveform ↴
v$="0" ↴
e$="0" ↴
w$="0" ↴
l$="0" ↴
d$="0" ↴
@wave_string ↴
PUT 455,90,r$ ↴
TEXT 460,99,"Note" ↴
@note_box ↴
@notes ↴
@sound_string ↴
' ↴
' Exit Button ↴
BOX 517,52,637,103 ↴
PBOX 518,53,636,102 ↴
DEFTEXT 1,5,,32 ↴
TEXT 540,86,"Exit" ↴
DEFTEXT 1,1,,6 ↴
RETURN ↴
' ↴
```

This procedure draws the selector buttons for the notes in the SOUND command.

```
PROCEDURE note_box ..  
' ..  
' Draw selectors for Notes ..  
FOR c%=0 TO 1 ..  
  FOR ctr%=0 TO 5 ..  
    PUT 330+ctr%*50,110+c%*15,g$ ..  
  NEXT ctr% ..  
NEXT c% ..  
RETURN ..  
' ..
```

Show the note values for the buttons.

```
PROCEDURE notes ..  
RESTORE note_data ..  
FOR x%=0 TO 1 ..  
  FOR y%=0 TO 5 ..  
    READ n$ ..  
    TEXT 343+y%*50,119+x%*15,n$ ..  
  NEXT y% ..  
NEXT x% ..  
note_data: ..  
DATA C,C#,D,D#,E,F,F#,G,G#,A,A#,B ..  
RETURN ..  
' ..
```

Draw the boxes for the wave forms, and draw in the graphic representation of each wave.

```
PROCEDURE wave_boxes ..  
' ..  
FOR c%=0 TO 1 ..  
  FOR ctr%=0 TO 4 ..  
    PUT 34+ctr%*50,149+c%*15,g$ ..  
  NEXT ctr% ..  
NEXT c% ..  
RETURN ..  
' ..
```

```
PROCEDURE waveform ↴
  ↴
  ' Draw wave forms ↴
  ↴
  ' Linear falling ↴
  DRAW 39,152 TO 46,158 TO 70,158 ↴
  DRAW 38,152 TO 45,158 ↴
  ↴
  ' Linear rising terminating
  DRAW 88,158 TO 99,152 TO 99,158 TO 120,158
  ↴

  ' Sawtooth Falling ↴
  DRAW 140,152 TO 150,158 TO 150,152 TO 160,158 TO 160,152 TO 170,158 ↴
  ↴
  ' Linear Falling ↴
  DRAW 190,152 TO 196,158 TO 220,158 ↴
  DRAW 191,152 TO 197,158 ↴
  ↴
  ' Peaked, begin falling ↴
  DRAW 237,153 TO 245,158 TO 253,153 TO 262,158 TO 271,153 ↴
  ↴
  ↴
  ' Linear falling, variable loudness ↴
  DRAW 38,168 TO 46,173 TO 46,168 TO 71,168 ↴
  ↴
  ' Sawtooth Rising ↴
  DRAW 88,173 TO 98,168 TO 98,173 TO 108,168 TO 108,173 TO 118,168 TO 118,173 ↴
  ↴
  ' Linear Rising Continuous ↴
  DRAW 140,173 TO 150,168 TO 170,168 ↴
  ↴
  ' Peaked, begin rising ↴
  DRAW 190,173 TO 195,168 TO 200,173 TO 205,168 TO 210,173 TO 215,168 TO 220,173 ↴
  ↴
  ' Linear Rising ↴
  DRAW 238,173 TO 250,168 TO 250,173 TO 271,173 ↴
  ↴
RETURN ↴
  ↴
```

Next draw the boxes which show the settings for WAVE and SOUND.

```

PROCEDURE wave_string ↴
  DEFFILL 3 ↴
  RBOX 1,177,317,199 ↴
  PRBOX 2,178,318,198 ↴
  DEFTEXT 2,0,,6 ↴
  TEXT 20,187,"WAVE voc, env, form, len, dur" ↴
  wave$="WAVE "+v$+","+e$+","+w$+","+l$+","+d$ ↴
  TEXT 20,196,wave$ ↴
  DEFTEXT 1,1,,6 ↴
RETURN ↴
' ↴
PROCEDURE sound_string ↴
  RBOX 324,158,639,199 ↴
  PRBOX 325,159,638,198 ↴
  DEFTEXT 2,0,,6 ↴
  TEXT 340,166,"SOUND voc, vol, note, oct, dur" ↴
  @build_sound ↴
  TEXT 340,176,s1$ ↴
  TEXT 340,186,s2$ ↴
  TEXT 340,196,s3$ ↴
  DEFTEXT 1,1,,6 ↴
RETURN ↴
' ↴
PROCEDURE build_sound ↴
  ' build strings for SOUND display ↴
  s1$="SOUND 1" + "," + STR$(vol%(1))+"," + STR$(n%(1)) + "," + STR$(oct%(1)) + "," + STR$(dur%(1)) ↴
  s2$="SOUND 2" + "," + STR$(vol%(2))+"," + STR$(n%(2)) + "," + STR$(oct%(2)) + "," + STR$(dur%(2)) ↴
  ↴
  s3$="SOUND 3" + "," + STR$(vol%(3))+"," + STR$(n%(3)) + "," + STR$(oct%(3)) + "," + STR$(dur%(3)) ↴
RETURN ↴
' ↴

```

The next two procedures have been discussed before. For more information, refer to Chapter 3. (One of the things I hate is a program which changes my screen settings, then ends without cleaning up, leaving me with an unreadable screen, forcing me to waste time rebooting.)

```
PROCEDURE save_palette ..  
LOCAL ctr% ..  
FOR ctr%=0 TO 15 ..  
    palette%(ctr%)=XBIOS(7,W:ctr%,W:-1) ..  
NEXT ctr% ..  
RETURN ..  
' ..  
PROCEDURE restore_palette ..  
SHOWM ..  
LOCAL ctr% ..  
FOR ctr%=0 TO 15 ..  
    SETCOLOR ctr%,palette%(ctr%) ..  
NEXT ctr% ..  
RETURN ..
```

Sound Edit is a simple sound processor. With a little more effort, a routine could have been added to generate code in ASCII format and write it to disk. Since the command syntax for SOUND and WAVE is relatively easy to understand, this step was omitted.

Write down the parameters displayed in the SOUND and WAVE boxes by your experiments, and just patch these parameters into your original code.

If you'd like, study the code generation sequences in the Shape Editor in Chapter 4, and write your own code generator for Sound Edit.



Chapter 6

Input/Output



Chapter 6 Input/Output

Communicating with other devices, such as disk drives, printers, and modems is an often frustrating task for many novice programmers. One of the nicest features of GFA BASIC is the ease with which these tasks can be accomplished.

This chapter will demonstrate the efficiency of Input/Output functions with a rather long demonstration program. Although this program may not suit your needs at this time, take a look at the many procedures which address various I/O devices of your computer.

Telecommunications

One of the more interesting aspects of operating a computer is the ability to communicate with other computers. To the novice, this seems a strange and mysterious practice, suitable only for "hackers" intent on accessing restricted material.

Of course, this negative image can be blamed on the news media and movies.

Actually thousands of times every day people just like you are using their computers to call other computers. The computers being called may be the huge banks of computers utilized by the public information services, such as GEnie (the General Electric Network for Information Exchange), a privately owned and operated bulletin board, or a computer at their office.

People call computers for a variety of reasons. Some call to obtain the countless public domain programs which are available. Others to participate in the "give and take" of the on-line discussions. Others, to meet new people through live, real time on-line conferencing.

You can use services such as GEnie to make airplane and hotel reservations, to shop at home, catch up on the latest news from USA Today, monitor prices of stocks and bonds, or even access information from on-line encyclopedias. There are as many reasons for computer communications as there are people communicating via their computers.

One word of warning: some private BBS operators are using their computers to illegally distribute copyrighted software. Often people refer to these bulletin boards as "Pirate" boards.

These operators are breaking the law. Many software developers are taking aggressive action against such operators. Recent court cases have awarded cash damages and computer equipment to the developers.

Usually, all records, including the user log, from the BBS are also seized. If your name appears in this log *YOU* could be investigated for possible criminal activity.

If you become aware of illegal activity on a BBS of which you are a participant, we advise you to stop visiting that board, and report such activity to either the software developer, or to other proper authorities. Recently, records of one such BBS were turned over to Federal authorities for further investigation of possible fraud.

Most BBSs are run by honest individuals, or computer clubs, so we hope you'll participate in legal activities, and enjoy the new horizons of communication, via computers.

BASIC Communication

Many people feel that telecommunication is an extremely difficult practice, and that there's some "magic" involved in writing an effective terminal program.

When you telecommunicate, you are, in essence, turning your computer into a terminal, and operating the host systems computer. All that's necessary, at the most basic level, is to send the characters you type to the host system, and display the information received from the host on your screen.

With GFA BASIC, the process is quite simple. Program 6-1 is a "dumb" terminal program. It doesn't process incoming data. Also it can't save data to disk, or upload and download files from disk.

It's nothing fancy, but it's the nucleus of any communications program. It just communicates.

Program 6-1 Dumb Terminal

For more information about the parameters for the XBIOS #15, RSCONF call used in this program, refer to Appendix C, XBIOS Functions, page C-14.

```
CLS ↴
' ↴
' Configure RS-232 using XBIOS #15, rsconf ↴
' ↴
' ↴
rsconf=15 ↴
baud=7           ! 1200 Baud ↴
flow=0 ↴
ucr=&X10101110    ! decimal 174 ↴
rsr=-1 ↴
tsr=-1 ↴
scr=-1 ↴
'
VOID XBIOS(rsconf,baud,flow,ucr,rsr,tsr,scr) ↴
'
```

After the RS-232 port has been configured, just open a data channel for the RS-232 port.

This terminal program runs in half duplex mode so all key strokes are echoed to the screen.

```
OPEN "U",#1,"AUX:" ↴
```

Now, here's the "action loop". Actually, this is the entire program. Check the keyboard for a key press, if a key has been pressed, send the ASCII value to the RS-232 port.

```
DO ↴
  ' ↴
  EXIT IF ASC(k$)=27      ! Escape Key pressed ↴
  k$=INKEY$ ↴
  IF k$<>"" ↴
    OUT 1,ASC(k$) ↴
    IF ASC(k$)=13 ↴
      OUT 2,10 ↴
    ENDIF ↴
  ENDIF ↴
  ' ↴
```

Check the RS-232 port for an incoming character. If a character is present, display it on the screen.

```
IF INP?(1) ↴
  ' ↴
  a=INP(1) ↴
  OUT 2,a ↴
ENDIF ↴
' ↴
LOOP ↴
' ↴
```

The escape <Esc> is the exit key for the program. IF <Esc> is pressed, exit the main loop, and close the data channel.

```
CLOSE #1 ↴
```

Getting Fancy

As further proof that GFA BASIC is an extremely powerful language, Program 6-2 is a full-featured terminal program. It has all the features found in most terminal programs, and a few extras thrown in for convenience.

Even though this is a terminal program, you'll notice that the majority of the program is the user interface. This user interface is common to any program. As a programmer, your goal is to create a program which is as easy to use as possible. Whenever you can, try to make your programs intuitive. That is, make any series of actions required to implement a process "feel" natural.

Program 6-2 EasyTerm

EasyTerm is copyrighted by MICHTRON, Inc., as are all the programs in this book. You, by purchasing a copy of this book, are entitled to a copy of this program, and may modify and use it as you see fit. However, we ask that you respect our copyright, and do not distribute it to users which have not purchased a copy of the *GFA BASIC Programmers Reference Guide, Vol. I*.

This is the longest, and most involved program in this book. Please be careful typing it in. All lines must be entered *exactly* as listed.

Remember that the indentations need not be entered, as the GFA BASIC Editor/Interpreter will automatically indent where needed.

EasyTerm begins by reserving memory for variable arrays.

```
'  
' EasyTerm ..  
' Written in GFA BASIC by George W. Miller ..  
' Copyright 1988 MichTron, Inc. ..  
' All Rights Reserved ..  
'  
DIM strip$(93),buf(127),number$(9),service$(9),key$(19) ..  
DIM palette(15),pass$(9),prompt$(9) ..  
DIM script$(100) ..  
'  
'  
'
```

Next, as the program goes through its initialization process, the shape of the mouse is changed to the "Bee" to inform the user that something is happening. Also a 32K block of memory is reserved because this program will be using alert boxes, and the fileselector.

```
DEFMOUSE 2 .  
CLS .  
RESERVE FRE(0)-32767 .
```

Now some default constants are defined. The default settings, installed as the program boots, are for GEnie.

```
' .  
' Configure RS-232 using XBIOS #15, rsconf, using following defaults: .  
' .  
capture=0 .  
rsconf=15 .  
flow=0 .  
ucr=&X10001000 .  
' UCR default settings:      GEnie defaults .  
'   no parity , 1 stop bit, 8 bits .  
' .  
rsr=-1 .  
tsr=-1 .  
scr=-1 .  
full=0 .  
half=1 .  
baud=7 .  
' .
```

Xmodem routines will be available for data transfer, so define some constants for use by the Xmodem procedure.

```
' definitions for XMODEM ..  
' ..  
soh=1      ! Start Of Header ..  
eot=4      ! End Of Transmission ..  
ack=6      ! ACKnowledge ..  
nak=21     ! Negative ACKnowledge ..  
can=24     ! Cancel ..  
xon=17 ..  
xoff=19 ..  
dial_now=0 ..  
' ..
```

Although this system poke is not really necessary, it does speed up disk access by a small amount.

```
SPOKE &H444,0           ! Turn off Write Verify ..
```

Save the path name. If the program was booted from within a folder, this will enable the support files to be located within the folder.

```
drive=GEMDOS(&H19) ..  
path_name$=SPACE$(64)          ! Reserve space for path_name ..  
' ..  
ptr=VARPTR(path_name$)        ! Define pointer to variable ..  
' ..  
' Call GEMDOS function $47, D_getpath ..  
' Pass the pointer to the buffer to hold the path name with a long word, ..  
' and pass the drive to be used with a word. ..  
' ..  
VOID GEMDOS(&H47,L:ptr,W:drive+1) ..
```

```
' ↴
' GEMDOS $47 returns the name at the address pointed to with a null (0) byte ↴
' as the terminator. Search for this null byte, and define path_name$ ↴
' for fileselector. ↴
' ↴
FOR ctr%=1 TO 63 ↴
  IF ASC(MID$(path_name$,ctr%,1))=0 ↴
    path_name$=LEFT$(path_name$,ctr%-1) ↴
    ctr%=63 ↴
  ENDIF ↴
NEXT ctr% ↴
' ↴
```

Here are the data elements which will be used for the menu. Be careful to type in spaces as needed to align the data items in the menu.

```
DATA Desk, EasyTerm ↴
DATA ----- ↴
DATA 1,2,3,4,5,6,"" ↴
DATA Baud, 19200, 9600, 4800, 3600, 2400, 2000 ↴
DATA 1800, 1200, 600, 300, 150, 134 ↴
DATA 110, 75, 50,"" ↴
DATA RS-232, Full Duplex, Half Duplex, 8 Bit Word, 7 Bit Word ↴
DATA No Parity, Odd Parity, Even Parity, 1 Stop Bit, 2 Stop Bits ↴
DATA -----, Save RS232.CFG, Load .CFG file ,"" ↴
DATA Files, UL/ASCII, UL/Xmodem, DL/ASCII, DL/Xmodem ↴
DATA -----, Buffer OPEN, Buffer CLOSED, SAVE Buffer ↴
DATA CLEAR Buffer, Print Buffer ↴
DATA -----, Load Buffer, View Buffer ↴
DATA -----, Execute File,"" ↴
DATA DOS, Free Space, Create Folder, Erase File, Rename File ↴
DATA Directory, -----, Disk A, Disk B, Disk C ↴
DATA Disk D, Disk E, Disk F,"" ↴
DATA Utilities, Auto Dial, -----, Printer ON, Printer OFF ↴
DATA -----, Set Time/Date ↴
DATA -----, Define Function Keys ↴
DATA -----, Disk Verify ON, Disk Verify OFF ↴
```

```
DATA -----, Script File ↴
DATA -----, Help ,"" ↴
DATA Exit, Quit ,"" ↴
DATA *** ↴
' ↴
```

As you can see, the menu is quite extensive.

Since one of the menu selections is for the disk drives, check to see which drives are actually connected and available to the program by calling BIOS(10).

```
dr_map=BIOS(10)
```

This program won't look very good in low resolution, so check for medium or high resolution before continuing.

```
rez=XBIOS(4) ↴
IF rez=0 ↴
  alrt$=" EasyTerm| requires Medium|or High Resolution" ↴
  ALERT 3,alrt$,1,"OK",b ↴
  END ↴
ENDIF ↴
' ↴
```

Again, save the default palette settings so they can be restored when the program ends, then read the menu items to build the menu array.

```
@save_palette ↴
ctr%=0 ↴
DO ↴
  READ strip$(ctr%) ↴
  EXIT IF strip$(ctr%)="***" ↴
  INC ctr% ↴
LOOP ↴
strip$(ctr%)="" ↴
strip$(ctr%+1)="" ↴
ON MENU GOSUB menu ↴
' ↴
```

Here are a few more program constants.

```
echo=half      !default to Half duplex mode ..  
buf=on ..
```

With the initialization delays now over, redefine the mouse to a pointer, and display the alert box used for the title sequence.

```
DEFMOUSE 0 ..  
' ..  
alrt$="EasyTerm Version 1.0 |Written in GFA BASIC| By George Miller| May 1988" ..  
ALERT 0,alrt$,1,"OK",b ..  
' ..
```

Change the mouse pointer to a "bee" again, and check for any presaved configuration files. If any are found, check if the user would like to install it.

The procedure have_file is a special routine to check for files on a disk. The GFA BASIC EXIST function could also have been used.

```
DEFMOUSE 2 ..  
filename$=path_name$+"\*.CFG" ..  
@have_file ..  
IF is_it=0 ..  
  DEFMOUSE 0 ..  
  alrt$=" EasyTerm| |Load RS232.CFG?" ..  
  ALERT 2,alrt$,1,"Yes|No",b ..  
  IF b=1 ..  
    CLS ..  
    @load_cfg ..  
  ENDIF ..  
ENDIF ..  
' ..
```

Check for a phone directory, as above.

```
filename$=path_name$+"\\*.PHN" ↴
@have_file ↴
IF is_it=0 ↴
  DEFMOUSE 0 ↴
  alrt$=" EasyTerm| |Load Autodial Directory?" ↴
  ALERT 2,alrt$,1,"Yes|No",b ↴
  ↴
  IF b=2 ↴
    GOTO auto_fini ↴
  ENDIF ↴
  ↴
  auto_load: ↴
  CLS ↴
  PRINT "Select Auto Dial File to load:" ↴
  FILESELECT CHR$(drive+65)+":\"+path_name$+"\\*.phn",b$,filename$ ↴
  DEFMOUSE 2 ↴
  IF filename$="" ↴
    GOTO auto_fini ↴
  ENDIF ↴
  @have_file ↴
```

If the file specified is not on the disk, a value of -33 is returned. This routine informs the user of a problem, and offers a chance to correct it.

```
IF is_it=-33 ↴
  DEFMOUSE 0 ↴
  alrt$=" EasyTerm| |Can't find that file." ↴
  ALERT 1,alrt$,1,"OK",b ↴
  GOTO auto_load ↴
ENDIF ↴
  ↴
  CLS ↴
  IF filename$="" ↴
    GOTO auto_fini ↴
  ENDIF ↴
  ↴
```

If there is a file, open it and load the variables with the proper values.

```
OPEN "I",#3,filename$ ..  
ctr%=0 ..  
FOR ctr%=0 TO 9 ..  
    LINE INPUT #3,service$(ctr%);number$(ctr%);pass$(ctr%);prompt$(ctr%) ..  
NEXT ctr% ..  
CLOSE #3 ..  
auto_fini: ..  
b$="" ..  
ENDIF ..  
' ..
```

Again, a check for a file containing predefined function keys.

```
filename$=path_name$+"funct.key" ..  
@have_file ..  
IF is_it=0 ..  
    DEFMOUSE 0 ..  
    alrt$="    EasyTerm| |Load Defined Function keys?" ..  
    ALERT 2,alrt$,1,"Yes|No",b ..  
    ' ..  
    IF b=1 ..  
        DEFMOUSE 2 ..  
        PRINT "Loading Pre-defined Function Keys." ..  
        OPEN "I",#3,"funct.key" ..  
        ' ..  
        FOR ctr%=0 TO 19 ..  
            LINE INPUT #3;key$(ctr%) ..  
        NEXT ctr% ..  
        ' ..  
        CLOSE #3 ..  
        DEFMOUSE 0 ..  
        ' ..  
        CLS ..  
    ENDIF ..  
ENDIF ..  
' ..
```

A few more preliminary routines are called, and a data channel is opened to the RS-232 port.

```
DEFMOUSE 0 ↴
@check_it ↴
@rsconf ↴
HIDEM ↴
OPEN "U",#1,"AUX:" ↴
' ↴
```

The program actually begins to work now. All background operations are performed and necessary checks are made.

```
@menu ↴
CLS ↴
DO ↴
' ↴
IF colour=1 ↴
  @colour ↴
ENDIF ↴
IF dial_now=1 ↴
  @dial_it ↴
ENDIF ↴
IF MOUSEK=1 ↴
  @check_it ↴
  HIDEM ↴
ENDIF ↴
```

The call to XBIOS #21 turns on the cursor, and the program begins checking for keyboard characters, and the RS-232 port.

```
VOID XBIOS(21,1) ↴
IF INP?(1)<>0 ↴
  t=INP(1) ↴
  IF t=8 ↴
    OUT 2,32 ↴
```

```
ENDIF ↴
OUT 2,t AND 127 ↴
' ↴
```

If the printer was been selected, *prnt!* is TRUE, so the byte stored in *t* is sent to the printer.

```
IF prnt!=-1 ↴
  OUT 0,t ↴
  IF t=13 ↴
    INC prnt_cnt ↴
    IF prnt_cnt=80 ↴
      OUT 0,13 ↴
      OUT 0,10 ↴
      prnt_cnt=0 ↴
    ENDIF ↴
  ENDIF ↴
ENDIF ↴
' ↴
```

If the capture buffer has been opened, a copy of the byte is also stored at the proper location in the capture buffer.

```
IF capture! ↴
  @capture_it ↴
ENDIF ↴
' ↴
ENDIF ↴
```

This next segment gets a little fancy. If a color monitor is being used, anything typed on the keyboard will be displayed in red on the terminal screen. Changing the color is taken care of by sending the VT-52 Terminal Escape Code, CHR\$(27);“b”. All of the defined keyboard functions are also checked. The called procedures will be discussed in detail later.

```
IF INP?(2)<>0 ↴
  IF rez=1 ↴
    PRINT CHR$(27);“b”;CHR$(1); ↴
```

```
ENDIF ↴
t=INP(2) ↴
IF t>=187 AND t<=196      !check function key ↴
    @function_key ↴
ENDIF ↴
IF t>=212 AND t<=22 !check shifted-function key ↴
    @sfuction_key ↴
ENDIF ↴
IF t=199 ↴
    CLS ↴
    t=0 ↴
ENDIF ↴
IF t>=150 AND t<=178 ↴
    @alt_key ↴
    GOTO no_send ↴
ENDIF ↴
IF t=226 ↴
    @help ↴
ENDIF ↴
OUT 1,t ↴
IF prnt!=-1 ↴
    OUT 0,t ↴
    INC prnt_cnt ↴
    IF prnt_cnt=80 ↴
        OUT 0,13 ↴
        OUT 0,10 ↴
        prnt_cnt=0 ↴
    ENDIF ↴
ENDIF ↴
' ↴
```

This segment takes care of processing the characters properly for Full and Half Duplex modes of operation. Also, the capture buffer is maintained.

```
IF t=8 AND echo=0 ↴
    OUT 2,8 ↴
    OUT 2,32 ↴
    OUT 2,8 ↴
ENDIF ↴
IF echo=1 ↴
```

```
IF t=8 ..  
    OUT 2,8 ..  
    OUT 2,32 ..  
ENDIF ..  
OUT 2,t AND 127 ..  
ENDIF ..  
IF capture! ..  
    @capture_it ..  
ENDIF ..  
ENDIF ..  
IF rez=1 ..  
PRINT CHR$(27);"b";CHR$(3); ..  
ENDIF ..  
' ..  
no_send: ..  
LOOP ..  
' ..
```

That's the end of the main loop of the program. In order to function properly, this loop must be kept as tight as possible. No unnecessary calls are permitted.

```
CLOSE #1 ..  
' ..  
' program should never get here, but if something goes wrong..... ..  
' ..  
END ..  
' ..
```

The next procedure takes care of the capture buffer. Characters received and transmitted are stored in a buffer until CHR\$(13), the ASCII code for a carriage return, is encountered, then the data is concatenated into capture\$. When the capture buffer reaches its maximum size, 32K bytes, the user is advised, and capture\$ may be appended to an existing buffer saved on the disk.

```
PROCEDURE capture_it ..  
' ..  
buffer$=buffer$+CHR$(t) ..  
IF t=13 ..  
    capture$=capture$+buffer$ ..
```

```
buffer$="" ↴
ENDIF ↴
INC capture ↴
IF capture=32766 ↴
```

Before interrupting the program, the host system is advised, by sending the constant xoff, to wait.

```
OUT 1,xoff ↴
capture$=capture$+buffer$+CHR$(13) ↴
buffer$="" ↴
capture!=0 ↴
SGET t$ ↴
' ↴
ALERT 2,"Save Capture buffer?",1,"YES|NO",b ↴
IF b=1 ↴
  OPEN "A",#1,"EASYTERM.BUF" ↴
  PRINT #1,capture$ ↴
  CLOSE #1 ↴
ENDIF ↴
capture$="" ↴
```

All processing done, send the constant xon to advise the host to continue sending.

```
OUT 1,xon ↴
ENDIF ↴
' ↴
RETURN ↴
' ↴
```

The next procedure allows procedures to be called from the output (terminal) screen by pressing keys in combination with the <Alternate> key. *t* is the ASCII value returned when a key is pressed. This value is determined in the main program loop.

Where appropriate, messages are displayed on the screen. EasyTerm changes the text color to blue for messages it sends in the terminal mode of operation.

```
PROCEDURE alt_key ↴
  ↴
  IF rez=1 ↴
    PRINT CHR$(27); "b";CHR$(2); ↴
  ENDIF ↴
  IF t=159 ↴
    d=0 ↴
    IF baud=4 ↴
      d=1 ↴
    ENDIF ↴
    IF baud=7 ↴
      d=2 ↴
    ENDIF ↴
    IF baud=9 ↴
      d=3 ↴
    ENDIF ↴
    alrt$=" |Set Baud rate to:" ↴
    ALERT 1,alrt$,d,"2400|1200|300",b ↴
    IF b=1 ↴
      baud=4 ↴
    ENDIF ↴
    IF b=2 ↴
      baud=7 ↴
    ENDIF ↴
    IF b=3 ↴
      baud=9 ↴
    ENDIF ↴
    @rsconf ↴
  ENDIF ↴
  IF t=178 ↴
    IF echo=half ↴
      echo=full ↴
      PRINT ↴
      PRINT CHR$(27); "p <FULL DUPLEX> ";CHR$(27); "q" ↴
      GOTO changed ↴
    ENDIF ↴
    IF echo=full ↴
      echo=half ↴
      PRINT ↴
      PRINT CHR$(27); "p <HALF DUPLEX> ";CHR$(27); "q" ↴
    ENDIF ↴
```

```
changed: ↴
ENDIF ↴
IF t=176 ↴
  IF capture!=-1 ↴
    capture!=0 ↴
    PRINT ↴
    PRINT CHR$(27);"p <CAPTURE BUFFER CLOSED>";CHR$(27);"q" ↴
    GOTO cap_changed ↴
  ENDIF ↴
  IF capture!=0 ↴
    capture!=-1 ↴
    capture=0 ↴
    PRINT ↴
    PRINT CHR$(27);"p <CAPTURE BUFFER OPEN>";CHR$(27);"q" ↴
  ENDIF ↴
  cap_changed: ↴
ENDIF ↴
IF t=174 ↴
  capture$="" ↴
  PRINT ↴
  PRINT CHR$(27);"p <CAPTURE BUFFER CLEARED>";CHR$(27);"q" ↴
ENDIF ↴
IF t=161 ↴
  drive=GEMDOS(&H19)+1 ↴
  disk_space=DFREE(drive) ↴
  PRINT ↴
  PRINT CHR$(27);"p <Free Space remaining on Disk ↴
";CHR$(drive+64);": ";disk_space;" Bytes.>";CHR$(27);"q" ↴
ENDIF ↴
IF t=153 ↴
  IF prnt!=0 AND prnt=-1 ↴
    prnt!=-1 ↴
    PRINT ↴
    PRINT CHR$(27);"p <PRINTER ON>";CHR$(27);"q" ↴
    GOTO changed_prnt ↴
  ENDIF ↴
  IF prnt!=-1 AND prnt=-1 ↴
    prnt!=0 ↴
    PRINT ↴
    PRINT CHR$(27);"p <PRINTER OFF>";CHR$(27);"q" ↴
  changed_prnt: ↴
```

```
ENDIF ↴
ENDIF ↴
IF t=172 ↴
  @buf_text ↴
ENDIF ↴
IF t=175 ↴
  IF PEEK(&H444)<>0 ↴
    SPOKE &H444,0 ↴
    PRINT ↴
    PRINT CHR$(27);"

<DISK VERIFY ON>";CHR$(27);"" ↴
  ELSE ↴
    SPOKE &H444,1 ↴
    PRINT ↴
    PRINT CHR$(27);"

<DISK VERIFY OFF>";CHR$(27);"" ↴
  ENDIF ↴
ENDIF ↴
IF t=150 ↴
  xcol=CRSCOL ↴
  ylin=CRSLIN ↴
  alrt$=" EasyTerm|Upload File|Select Protocol to use: " ↴
  ALERT 1,alrt$,3,"Xmodem|ASCII|Cancel",b ↴
  IF b=1 ↴
    @xmodem_send ↴
  ENDIF ↴
  IF b=2 ↴
    @ul_ascii ↴
  ENDIF ↴
  PRINT AT(xcol,ylin); ↴
ENDIF ↴
IF t=160 ↴
  xcol=CRSCOL ↴
  ylin=CRSLIN ↴
  alrt$=" EasyTerm|Download File|Select Protocol to use: " ↴
  ALERT 1,alrt$,3,"Xmodem|ASCII|Cancel",b ↴
  IF b=1 ↴
    @xmodem_receive ↴
  ENDIF ↴
  IF b=2 ↴
    @dl_ascii ↴
  ENDIF ↴
  PRINT AT(xcol,ylin); ↴


```

```
ENDIF ↴
IF t=173 ↴
  PRINT ↴
  @status ↴
ENDIF ↴
t=0 ↴
IF rez=1 ↴
  PRINT CHR$(27); "b";CHR$(3); ↴
  PRINT CHR$(27); "q"; ↴
ENDIF ↴
RETURN ↴
' ↴
```

The next two procedures define the screen colors for the terminal screen (black background) and the Menu screen (white background).

```
PROCEDURE colour ↴
  SETCOLOR 0,0,0 ↴
  SETCOLOR 1,7,0,0 ↴
  SETCOLOR 2,0,0,7 ↴
  SETCOLOR 3,7,7,7 ↴
  colour=0 ↴
RETURN ↴
' ↴

PROCEDURE alt_colour ↴
  SETCOLOR 0,7,7,7 ↴
  SETCOLOR 1,7,7,7 ↴
  SETCOLOR 2,0,0,7 ↴
  SETCOLOR 3,0,0,0 ↴
  colour=1 ↴
RETURN ↴
' ↴
```

Sometimes, especially when participating in live on-line conferences, it's handy to be able to isolate what you're typing from what other people are sending. This procedure opens an area on the screen and permits you to type your message, without having it scroll away, mixed in with other people's text.

```
PROCEDURE buf_text ..  
' ..  
OUT 1,19 ..  
b_text$="" ..
```

XBIOS &H15 hides the cursor while a copy of the screen is saved, then restores it.
The cursor is positioned on the screen by CRSCOL and CRSLIN.

```
VOID XBIOS(&H15,0) ..  
SGET screen$ ..  
VOID XBIOS(&H15,2) ..  
org_col=CRSCOL ..  
org_row=CRSLIN ..  
b_text$="" ..  
a=0 ..  
FOR ctr%=22 TO 24 ..  
    PRINT AT(ctr%,0);SPACE$(80); ..  
NEXT ctr% ..  
RBOX 1,181,639,195 ..  
IF rez=1 ..  
    PRINT CHR$(27);";b";CHR$(1); ..  
ENDIF ..  
PRINT AT(2,24);"> ";CHR$(27);";q"; ..  
trow=24 ..  
tcol=4 ..
```

This is the type ahead loop. The program stays here, until the user strikes the
<Return> key.

```
DO ..  
EXIT IF a=13 ..  
a=INP(2) ..  
IF a=8 ..  
    a=0 ..  
    DEC tcol ..  
    IF tcol<4 ..  
        tcol=4 ..
```

```
ENDIF ↴
PRINT AT(tcol,trow);" " ↴
PRINT AT(tcol,trow); ↴
IF LEN(b_txt$)<>0 ↴
    b_txt$=LEFT$(b_txt$,LEN(b_txt$)-1) ↴
ENDIF ↴
ENDIF ↴
IF a<>0 ↴
    b_txt$=b_txt$+CHR$(a) ↴
    PRINT AT(tcol,trow);CHR$(a); ↴
    INC tcol ↴
    IF tcol=79 ↴
        PRINT AT(2,24);>";CHR$(27);q"+SPACE$(76) ↴
        tcol=4 ↴
    ENDIF ↴
ENDIF ↴
LOOP ↴
IF b_txt$="" ↴
    GOTO no_txt ↴
ENDIF ↴
FOR ctr%=1 TO LEN(b_txt$) ↴
    OUT 1,ASC(MID$(b_txt$,ctr%,1)) ↴
NEXT ctr% ↴
t=13 ↴
' ↴
no_txt: ↴
VOID XBIOS(&H15,0) ↴
SPUT screen$ ↴
PRINT AT(org_col,org_row);"" ↴
VOID XBIOS(&H15,2) ↴
IF echo=1 ↴
    PRINT b_txt$ ↴
ENDIF ↴
IF rez=1 ↴
    PRINT CHR$(27);b";CHR$(3); ↴
ENDIF ↴
OUT 1,17 ↴
RETURN ↴
```

This procedure is called when the user leaves the terminal screen. The menu screen is displayed, and all menu functions are available to the user.

```
'  
PROCEDURE check_it()  
'  
flag_it=0  
x%=CRSCOL  
y%=CRSLIN  
VOID XBIOS(21,0)  
SGET screen1$  
CLS  
OPENW 0  
MENU strip$()  
@remove_check  
@menu_check  
SHOWM  
OUT 1,xoff  
@rt_btn  
DO  
  VSYNC  
  IF MOUSEK=2  
    flag_it=1  
  ENDIF  
  ON MENU  
  EXIT IF flag_it=1  
LOOP  
'  
OUT 1,xon  
CLOSEW 0  
SPUT screen1$  
'  
PRINT AT(x%,y%)  
HIDEM  
'  
RETURN  
'
```

Check marks are displayed on the menu. This routine is called to monitor the status of EasyTerm, and place the check marks at the correct positions.

```
PROCEDURE menu_check ↴
' ↴
IF colour=0 ↴
  @alt_colour ↴
ENDIF ↴
IF PEEK(&H444)=1 ↴
  MENU 82,0 ↴
  MENU 83,1 ↴
ENDIF ↴
IF PEEK(&H444)=0 ↴
  MENU 82,1 ↴
  MENU 83,0 ↴
ENDIF ↴
pmt=BIOS(8,0) ↴
IF pmt=0 ↴
  MENU 51,2 ↴
  MENU 75,2 ↴
  MENU 76,2 ↴
  pmt!=0 ↴
ENDIF ↴
IF pmt=-1 ↴
  MENU 51,3 ↴
  MENU 75,3 ↴
  MENU 76,3 ↴
ENDIF ↴
' ↴
IF pmt!=-1 AND pmt<>0 ↴
  MENU 75,1 ↴
ENDIF ↴
' ↴
IF pmt!=0 AND pmt<>0 ↴
  MENU 76,1 ↴
ENDIF ↴
MENU baud+11,1 ↴
IF echo=full ↴
  MENU 28,1 ↴
ELSE ↴
```

```
MENU 29,1 ↴
ENDIF ↴
IF ucr AND &X100000      !test word length ↴
  MENU 31,1 ↴
  MENU 30,0 ↴
ELSE ↴
  MENU 30,1 ↴
  MENU 31,0 ↴
ENDIF ↴
IF ucr AND &X100          !test parity settings ↴
  IF ucr AND &X10          !test parity (even) ↴
    MENU 34,1 ↴
    MENU 33,0 ↴
    MENU 32,0 ↴
  ELSE ↴
    MENU 33,1 ↴
    MENU 34,0 ↴
    MENU 32,0 ↴
  ENDIF ↴
ELSE ↴
  MENU 33,0 ↴
  MENU 34,0 ↴
  MENU 32,1 ↴
ENDIF ↴
' ↴
IF ucr AND &X11000      !test stop bits ↴
  MENU 35,1 ↴
  MENU 36,0 ↴
ENDIF ↴
' ↴
IF ucr AND &X10000 ↴
  MENU 36,1 ↴
  MENU 35,0 ↴
ENDIF ↴
' ↴
IF capture!=-1 ↴
  MENU 47,1 ↴
  MENU 48,0 ↴
ENDIF ↴
IF capture!=0 ↴
  MENU 47,0 ↴
```

```
MENU 48,1
ENDIF ..
'..
'
FOR ctr%=65 TO 70 ..
  MENU ctr%,2          !not selectable ..
NEXT ctr% ..
'..
FOR ctr%=65 TO 70 ..
  MENU ctr%,0          !not checked ..
NEXT ctr% ..
'..
IF dr_map AND &X11 ..
  MENU 65,3  !Drive A ..
  MENU 66,3  !Drive B ..
ENDIF ..
'..
IF dr_map AND &X100 ..
  MENU 67,3  !Drive C ..
ENDIF ..
'..
IF dr_map AND &X1000 ..
  MENU 68,3  !Drive D ..
ENDIF ..
'..
IF dr_map AND &X10000 ..
  MENU 69,3  !Drive E ..
ENDIF ..
'..
IF dr_map AND &X100000 ..
  MENU 70,3  !Drive F ..
ENDIF ..
'..
drive=GEMDOS(&H19) ..
MENU 65+drive,1          !Place check mark on active drive ..
PRINT AT(1,1); ..
@status ..
RETURN ..
'
```

Here's the procedure which takes care of all menu functions. It is called by the ON MENU statement earlier in the program.

This is an extremely long procedure, controlling all menu functions. If you are typing in this program, please be careful to include all spaces as indicated in the strings, or the menu functions will not be found.

```
PROCEDURE menue ..
```

The first menu function is "About EasyTerm", and displays an alert box with program information.

```
IF strip$(MENU(0))=strip$(1) ..  
  MENU OFF ..  
  alrt$=" EasyTerm Version 1.0| By George Miller"+CHR$(&HBD)+" 1988 MichTron, Inc.| All Rights  
  Reserved" ..  
  ALERT 0,alrt$,1," OK ",b ..  
 ENDIF ..
```

Next, the RS-232 configuration may be changed.

```
IF MENU(0)=30      !Set word length to 8 bits ..  
  MENU OFF ..  
  ucr=ucr AND &X10010000 ..  
  @menu_check ..  
  @rsconf ..  
ENDIF ..  
  
IF MENU(0)=31      !Set word length to 7 bits ..  
  MENU OFF ..  
  ucr=ucr OR &X10111000 ..  
  @menu_check ..  
  @rsconf ..  
ENDIF ..  
  
IF MENU(0)=32      !Set parity to none ..
```

```
MENU OFF ↴
ucr=ucr AND &X11111000 ↴
@menu_check ↴
@rsconf ↴
ENDIF ↴
' ↴
IF MENU(0)=33      !Set parity to odd ↴
  MENU OFF ↴
  ucr=ucr AND &X11111000 ↴
  ucr=ucr OR &X100 ↴
  @menu_check ↴
  @rsconf ↴
ENDIF ↴
' ↴
IF MENU(0)=34      !Set parity to even ↴
  MENU OFF ↴
  ucr=ucr AND &X11111000 ↴
  ucr=ucr OR &X110 ↴
  @menu_check ↴
  @rsconf ↴
ENDIF ↴
' ↴
IF MENU(0)=35      ! 1 Stop bit ↴
  MENU OFF ↴
  ucr=ucr AND &X11101110 ↴
  ucr=ucr OR &X1000 ↴
  @menu_check ↴
  @rsconf ↴
ENDIF ↴
' ↴
IF MENU(0)=36      ! 2 stop bits ↴
  MENU OFF ↴
  ucr=ucr AND &X11100110 ↴
  ucr=ucr OR &X11000 ↴
  @menu_check ↴
  @rsconf ↴
ENDIF ↴
' ↴
IF MENU(0)>10 AND MENU(0)<27  ! baud rate ↴
  MENU OFF ↴
  @baud_rate ↴
```

```
@rsconf ..  
' ..  
ENDIF ..  
' ..
```

If you call the BBS using the configuration you just saved, you'll want to save it as a .CFG file which may be installed by selecting and loading rather than clicking on the menu selectors each time. Notice that the fileselector uses the full pathname for the file, which permits saving it to a specified folder.

As the process continues, appropriate alert boxes are displayed.

```
IF strip$(MENU(0))=" Save RS232.CFG " ..  
' ..  
MENU OFF ..  
' ..  
rs232_choose: ..  
CLS ..  
PRINT "Save file as:" ..  
b$="RS232.CFG" ..  
FILESELECT CHR$(65+drive)+":"+path_name$+"*.CFG",b$,filename$ ..  
@have_file ..  
IF is_it=0 ..  
alrt$="File Exists!" ..  
ALERT 1,alrt$,1,"Replace|Cancel",b ..  
IF b=2 ..  
    GOTO rs232_done ..  
ENDIF ..  
ENDIF ..  
' ..  
IF filename$="" ..  
    GOTO rs232_done ..  
ENDIF ..  
' ..  
CLS ..  
PRINT "Saving file as: ",filename$ ..  
DEFMOUSE 2 ..  
OPEN "O",#3,filename$ ..  
' ..  
PRINT #3;flow ..  
PRINT #3;ucr ..
```

```
PRINT #3;sr ..  
PRINT #3;sr ..  
PRINT #3;scr ..  
PRINT #3;duplex ..  
PRINT #3;baud ..  
' ..  
CLOSE #3 ..  
DEFMOUSE 0 ..  
rs232_done: ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
' ..  
ENDIF ..  
' ..
```

If a configuration file has been saved, there must be a way to install it.

```
IF strip$(MENU(0))=" Load .CFG file " ..  
MENU OFF ..  
CLS ..  
@load_cfg ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
ENDIF ..  
' ..
```

Provide a means for an orderly end to the program.

```
IF strip$(MENU(0))=" Quit " ..  
MENU OFF ..  
IF capture!=-1 AND capture>0 ..  
alrt$="|Would you like to|Save the Text Buffer?" ..  
ALERT 2,alrt$,2," Yes | No",b ..  
IF b=1 ..  
b$="capture.txt" ..
```

```
FILESELECT CHR$(drive+65)+":"+path_name$+"\\*.TXT",b$,filename$ ..  
@have_file ..  
IF is_it=0 ..  
  alrt$="File Exists!" ..  
  ALERT 1,alrt$,2,"Replace|Cancel",b ..  
  IF b=1 ..  
    filename$="" ..  
  ENDIF ..  
ENDIF ..  
IF filename$="" ..  
  GOTO no_sav ..  
ENDIF ..  
' ..  
DEFMOUSE 2 ..  
OPEN "O",#3,filename$ ..  
PRINT #3,capture$ ..  
CLOSE #3 ..  
DEFMOUSE 0 ..  
' ..  
no_sav: ..  
ENDIF ..  
ENDIF ..  
' ..  
alrt$="|Do you really want to Quit?" ..  
ALERT 3,alrt$,2," OK |Cancel",b ..  
IF b=1 ..  
  RESERVE FRE(0)+32767-255 ..  
  @restore_palette ..  
  EDIT ..  
ENDIF ..  
ENDIF ..  
' ..
```

Change Duplex modes.

```
IF strip$(MENU(0))=" Full Duplex " ↴
    MENU OFF ↴
    MENU MENU(0),1 ↴
    MENU (MENU(0)+1),0 ↴
    echo=full ↴
    CLS ↴
    PRINT AT(1,1) ↴
    @status ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Half Duplex " ↴
    MENU OFF ↴
    MENU MENU(0),1 ↴
    MENU (MENU(0)-1),0 ↴
    echo=half ↴
    CLS ↴
    PRINT AT(1,1) ↴
    @status ↴
ENDIF ↴
```

Uploading and downloading files is an important function of any terminal program. It should be handled as easily and efficiently as possible. Both ASCII and Xmodem file transfers are supported.

```
' ↴
IF strip$(MENU(0))=" DL/ASCII "      !Download ASCII file ↴
    MENU OFF ↴
    @dl_ascii ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" UL/ASCII "      !Download ASCII file ↴
    MENU OFF ↴
    @ul_ascii ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" DL/Xmodem" ↴
    MENU OFF ↴
```

```
@xmodem_receive ..  
ENDIF ..  
' ..  
IF strip$(MENU(0))=" UL/Xmodem " ..  
MENU OFF ..  
@xmodem_send ..  
ENDIF ..  
' ..
```

The next two menu functions access the status of the Capture buffer. The buffer may be open or closed.

```
IF strip$(MENU(0))=" Buffer OPEN " ..  
MENU OFF ..  
MENU (MENU(0)),1 ..  
MENU MENU(0)+1,0 ..  
capture!=-1 ..  
capture=0 ..  
CLS ..  
PRINT AT(1,1); ..  
@status ..  
ENDIF ..  
' ..  
IF strip$(MENU(0))=" Buffer CLOSED " ..  
MENU OFF ..  
MENU (MENU(0)-1),0 ..  
MENU MENU(0),1 ..  
capture!=0 ..  
CLS ..  
PRINT AT(1,1); ..  
@status ..  
ENDIF ..  
' ..
```

If a buffer has been stored in memory, the user may want to save, print, or erase it from time to time.

```
IF strip$(MENU(0))=" SAVE Buffer" ↴
    MENU OFF ↴
    b$="EASYTERM.BUF" ↴
    FILESELECT CHR$(drive+65)+":"+path_name$+"*.TXT",b$,filename$ ↴
    @have_file ↴
    IF is_it=0 ↴
        alrt$="File Exists!" ↴
        ALERT 1,alrt$,3,"Replace|Append|Cancel",b ↴
        IF b=1 ↴
            filename$="" ↴
        ENDIF ↴
    ENDIF ↴
    IF filename$="" ↴
        GOTO skip_sav ↴
    ENDIF ↴
    ' ↴
    DEFMOUSE 2 ↴
    IF b=2 ↴
        OPEN "A",#3,filename$ ↴
    ENDIF ↴
    IF b=1 ↴
        OPEN "O",#3,filename$ ↴
        PRINT #3,capture$ ↴
        CLOSE #3 ↴
    ENDIF ↴
    DEFMOUSE 0 ↴
    capture!=0 ↴
    capture$="" ↴
    capture=0 ↴
    ' ↴
    skip_sav: ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Print Buffer" ↴
    MENU OFF ↴
    check_prn: ↴
    prmt=BIOS(8,0) ↴
    IF prmt=1 ↴
        LPRINT capture$ ↴
    ENDIF ↴
    IF prmt=0 ↴
```

```
alrt$=" |Printer not ready." ↴
ALERT 1,alrt$,1,"Retry|Cancel",b ↴
IF b=1 ↴
  GOTO check_prn ↴
ENDIF ↴
ENDIF ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" CLEAR Buffer " ↴
  MENU OFF ↴
  capture$="" ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
ENDIF ↴
' ↴
```

The next menu selection permits another program to be run from within EasyTerm. This will be explained in more detail later.

```
IF strip$(MENU(0))=" Execute File" ↴
  MENU OFF ↴
  @runit ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
ENDIF ↴
' ↴
```

Now several of the built-in GFA BASIC functions are used to provide the user with useful information.

```
IF strip$(MENU(0))=" Directory " ↴
  MENU OFF ↴
  CLS ↴
  FILES ↴
  @rt_btn ↴
ENDIF ↴
```

```
' ↴
IF strip$(MENU(0))=" Free Space " ↴
  MENU OFF ↴
  drive=GEMDO$(&H19)+1      !get active drive ↴
  disk_space=DFREE(drive) ↴
  CLS ↴
  alrt$="Free Disk Space remaining| on Disk "+CHR$(drive+64)+": | "+STR$(disk_space)+" Bytes" ↴
  ALERT 1,alrt$,1,"OK",b ↴
  @rt_btn ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Create Folder " ↴
  MENU OFF ↴
  CLS ↴
  PRINT "Create Folder:" ↴
  FILESELECT CHR$(drive+65)+":\*.*",b$,folder$ ↴
  IF folder$="" ↴
    GOTO fini_folder ↴
  ENDIF ↴
  check$=RIGHT$(folder$,4) ↴
  IF LEFT$(check$)=". " ↴
    alrt$=" |Not valid name." ↴
    ALERT 1,alrt$,1,"OK",b ↴
    GOTO fini_folder ↴
  ENDIF ↴
  MKDIR folder$ ↴
' ↴
fini_folder: ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
  @rt_btn ↴
' ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Erase File " ↴
  MENU OFF ↴
```

```
CLS ↴
PRINT "Select File to Delete:" ↴
FILESELECT CHR$(drive+65)+":\*.*",b$,filename$ ↴
IF filename$="" ↴
  GOTO fini_erase ↴
ENDIF ↴
' ↴
@have_file ↴
IF is_it=-33 ↴
  alrt$="File doesn't exists!" ↴
  ALERT 1,alrt$,1,"OK",b ↴
  GOTO fini_erase ↴
ENDIF ↴
' ↴
CLS ↴
alrt$="Ready to DELETE|" + filename$ + "|from disk.|Are you sure?" ↴
ALERT 2,alrt$,2,"OK|Cancel",b ↴
IF b=2 ↴
  GOTO fini_erase ↴
ENDIF ↴
KILL filename$ ↴
' ↴
fini_erase: ↴
CLS ↴
PRINT AT(1,1) ↴
@status ↴
@rt_btn ↴
' ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Rename File " ↴
  ↴
  MENU OFF ↴
  CLS ↴
  PRINT "Old File name:" ↴
  ↴
  FILESELECT CHR$(drive+65)+":\*.*",b$,filename$ ↴
  IF filename$="" ↴
    GOTO fini_rename ↴
  ENDIF ↴
  ↴
```

```
@have_file ..  
IF is_it=-33 ..  
  alrt$="File doesn't exists!" ..  
  ALERT 1,alrt$,1,"OK",b ..  
  GOTO fini_rename ..  
ENDIF ..  
' ..  
oldname$=filename$ ..  
CLS ..  
PRINT "New File name:" ..  
' ..  
FILESELECT CHR$(drive+65)+":\*.*",b$,filename$ ..  
IF filename$="" ..  
  GOTO fini_rename ..  
ENDIF ..  
' ..  
@have_file ..  
IF is_it=0 ..  
  alrt$="File already exists!" ..  
  ALERT 1,alrt$,2,"Replace|Cancel",b ..  
  IF b=2 ..  
    GOTO fini_rename ..  
  ENDIF ..  
ENDIF ..  
nuname$=filename$ ..  
' ..  
alrt$="About to rename "+oldname$+" as "+nuname$ ..  
ALERT 1,alrt$,2,"OK|Cancel",b ..  
IF b=2 ..  
  GOTO fini_rename ..  
ENDIF ..  
NAME oldname$ AS uname$ ..  
' ..  
fini_rename: ..  
  CLS ..  
  PRINT AT(1,1) ..  
  @status ..  
  @rt_btn ..  
ENDIF ..  
' ..  
IF strip$(MENU(0))=" Set Time/Date" ..
```

```
' ↴
MENU OFF ↴
CLS ↴
date: ↴
' ↴
rez=XBIOS(4) ↴
' ↴
nu_date$=DATE$ ↴
' ↴
PRINT AT(3,6); "Date presently set to: "; ↴
PRINT nu_date$ ↴
PRINT AT(3,8); "Press <Return> to use this date or" ↴
PRINT AT(3,10); "Enter new date (mm/dd/yy) "; ↴
PRINT AT(3,12); "(Be sure to include the '/'") ↴
PRINT AT(14,20); ↴
FORM INPUT 8,in_date$ ↴
IF in_date$="" ↴
  GOTO time ↴
ENDIF ↴
' ↴
IF ASC(in_date$)<48 OR ASC(in_date$)>57 ↴
  GOTO date ↴
ENDIF ↴
nu_date$=in_date$ ↴
' ↴
time: ↴
' ↴
CLS ↴
PRINT AT(3,6); "Present time is "; ↴
PRINT TIME$ ↴
PRINT AT(3,8); "Press <Return> to accept time or" ↴
PRINT AT(3,10); "Enter new time (hh:mm:ss) "; ↴
PRINT AT(3,12); "(Use Military [24 hour] format)" ↴
PRINT AT(14,20); ↴
FORM INPUT 8,in_time$ ↴
' ↴
nu_time$=in_time$ ↴
IF in_time$="" ↴
  nu_time$=TIME$ ↴
ENDIF ↴
' ↴
```

```
SETTIME nu_time$,nu_date$ ..  
' ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
@rt_btn ..  
ENDIF ..  
' ..
```

The next section permits changing the default drive to any attached drive.
ONLY drives which are actually connected may be selected.

```
IF strip$(MENU(0))=" Disk A " ..  
MENU OFF ..  
@delete_check ..  
MENU 65,1 ..  
CHDRIVE 1 ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
ENDIF ..  
' ..  
IF strip$(MENU(0))=" Disk B " ..  
MENU OFF ..  
@delete_check ..  
MENU 66,1 ..  
CHDRIVE 2 ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
ENDIF ..  
' ..  
IF strip$(MENU(0))=" Disk C" ..  
MENU OFF ..  
@delete_check ..  
MENU 67,1 ..  
CHDRIVE 3 ..  
CLS ..  
PRINT AT(1,1) ..
```

```
@status ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Disk D " ↴
  MENU OFF ↴
  @delete_check ↴
  MENU 68,1 ↴
  CHDRIVE 4 ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Disk E " ↴
  MENU OFF ↴
  @delete_check ↴
  MENU 69,1 ↴
  CHDRIVE 5 ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Disk F " ↴
  MENU OFF ↴
  @delete_check ↴
  MENU 70,1 ↴
  CHDRIVE 6 ↴
  CLS ↴
  PRINT AT(1,1) ↴
  @status ↴
ENDIF ↴
' ↴
```

Turn on or off an attached printer.

```
IF strip$(MENU(0))=" Printer ON " ↴
  MENU OFF ↴
  IF baud<9 ↴
```

```
alrt$="On Line Printer requires|Print Spooler program|at higher Baud rates." ↴
ALERT 1,alrt$,2,"OK|Cancel",b ↴
IF b=2 ↴
    GOTO no_prnt ↴
ENDIF ↴
ENDIF ↴
MENU 75,1 ↴
MENU 76,0 ↴
prmt!=-1 ↴
no_prnt: ↴
CLS ↴
PRINT AT(1,1) ↴
@status ↴
ENDIF ↴
' ↴
IF strip$(MENU(0))=" Printer OFF" ↴
    MENU OFF ↴
    MENU 75,0 ↴
    MENU 76,1 ↴
    prmt!=0 ↴
    CLS ↴
    PRINT AT(1,1) ↴
    @status ↴
ENDIF ↴
' ↴
```

Access to autodial phone functions.

```
IF strip$(MENU(0))=" Auto Dial " ↴
' ↴
    MENU OFF ↴
    @auto_dial ↴
    PRINT AT(1,1) ↴
    @status ↴
' ↴
ENDIF ↴
' ↴
```

Next, provide a means for defining the function keys.

```
IF strip$(MENU(0))=" Define Function Keys" ..  
    MENU OFF ..  
    alrt$=" EasyTerm| Define Function Keys " ..  
    ALERT 1,alrt$,3,"F1-F10|SF1-SF10|Cancel",b ..  
    IF b=1 ..  
        @edit_function_key ..  
    ENDIF ..  
    IF b=2 ..  
        @edit_shiftfunction_key ..  
    ENDIF ..  
    IF b=3 ..  
        GOTO skip_define ..  
    ENDIF ..  
skip_define: ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
@rt_btn ..  
ENDIF ..  
' ..
```

Turn disk verify on and off by setting or clearing byte.

```
IF strip$(MENU(0))=" Disk Verify ON " ..  
    MENU OFF ..  
    SPOKE &H444,0 ..  
    MENU 82,1 ..  
    MENU 83,0 ..  
    CLS ..  
    PRINT AT(1,1) ..  
    @status ..  
ENDIF ..  
' ..  
IF strip$(MENU(0))=" Disk Verify OFF" ..  
    MENU OFF ..  
    SPOKE &H444,1 ..
```

```
MENU 82,0 ↴
MENU 83,1 ↴
CLS ↴
PRINT AT(1,1) ↴
@status ↴
ENDIF ↴
' ↴
```

The next menu selection provides access to the Script processor. This, too, will be explained later.

```
IF strip$(MENU(0))=" Script File" ↴
MENU OFF ↴
alrt$=" EasyTerm| Script File| Processor|" ↴
ALERT 1,alrt$,1,"OK",b ↴
@script ↴
CLS ↴
PRINT AT(1,1) ↴
@status ↴
@rt_btn ↴
ENDIF ↴
' ↴
```

This menu option opens a rather extensive help screen.

```
IF strip$(MENU(0))=" Help " ↴
MENU OFF ↴
@help ↴
@rt_btn ↴
CLS ↴
PRINT AT(1,1) ↴
@status ↴
ENDIF ↴
' ↴
```

Loads an ASCII file from disk into the capture buffer.

```
IF strip$(MENU(0))=" Load Buffer " ..
  MENU OFF ..
  CLS ..
  alrt$="Loading Capture Buffer will overwrite contents!" ..
  ALERT 1,alrt$,2,"OK|Cancel",b ..
  IF b=2 ..
    GOTO no_load ..
  ENDIF ..
  PRINT "Select File to Load:" ..
  FILESELECT CHR$(drive+65)+":"+path_name$+"*.*",b$,filename$ ..
  IF filename$="" ..
    GOTO no_load ..
  ENDIF ..
  ' ..
  @have_file ..
  IF is_it=-33 ..
    alrt$="File doesn't exists!" ..
    ALERT 1,alrt$,1,"OK",b ..
    GOTO no_load ..
  ENDIF ..
  ' ..
  CLS ..
  PRINT "Loading ";filename$ ..
  DEFMOUSE 2 ..
  OPEN "I",#3,filename$ ..
  capture$="" ..
  DO ..
    EXIT IF EOF(#3) ..
    INPUT #3,a$ ..
    capture$=capture$+a$+CHR$(10)+CHR$(13) ..
  LOOP ..
  CLOSE #3 ..
  DEFMOUSE 0 ..
  ' ..
  no_load: ..
  CLS ..
  PRINT AT(1,1) ..
  @status ..
```

```
@rt_btn ..  
' ..  
ENDIF ..  
' ..
```

Prints the capture buffer to the screen. The <Control><S> and <Control><Q> keys interrupt and restart scrolling.

```
IF strip$(MENU(0))=" View Buffer" ..  
    MENU OFF ..  
    CLS ..  
    IF LEN(capture$)=0 ..  
        alrt$=" |Buffer is empty" ..  
        ALERT 1,alrt$,1,"OK",b ..  
        GOTO fini_view ..  
    ENDIF ..  
    FOR ctr%=1 TO LEN(capture$) ..  
        PRINT MID$(capture$,ctr%,1); ..  
        IF INP?(2)<>0 ..  
            key=INP(2) ..  
            IF key=19 ..  
                REPEAT ..  
                UNTIL INP(2)=17 ..  
            ENDIF ..  
            IF key=27 ..  
                ctr%=LEN(capture$) ..  
            ENDIF ..  
        ENDIF ..  
    NEXT ctr% ..  
fini_view: ..  
PRINT ..  
PRINT "      <End of Buffer -- Press any key to continue>" ..  
REPEAT ..  
UNTIL INKEY$<>"" ..  
CLS ..  
PRINT AT(1,1) ..  
@status ..  
@rt_btn ..  
ENDIF ..
```

```
'  
RETURN  
'
```

Procedure menu was an extremely long procedure, but it handles all menu items. The following procedures are called by the menu procedure, or by pressing the keyboard keys, as appropriate.

PROCEDURE status displays the status board on the terminal or menu screens.

This code is not very efficient. A better method would have been to build arrays for the data items, then use a pointer to the proper array.

```
PROCEDURE status '  
'
```

```
PRINT '  
PRINT "Terminal Status:" '  
PRINT '  
PRINT "Duplex Mode: "; '  
IF echo=full '  
    PRINT "FULL" '  
ELSE '  
    PRINT "HALF" '  
ENDIF '  
PRINT "Baud Rate: "; '  
IF baud=0 '  
    PRINT "19200" '  
ENDIF '  
IF baud=1 '  
    PRINT " 9600" '  
ENDIF '  
IF baud=2 '  
    PRINT " 4800" '  
ENDIF '  
IF baud=3 '  
    PRINT " 3600" '  
ENDIF '  
IF baud=4 '  
    PRINT " 2400" '  
ENDIF '  
IF baud=5 '  
'
```

```
PRINT " 2000" ..
ENDIF ..
IF baud=6 ..
    PRINT " 1800" ..
ENDIF ..
IF baud=7 ..
    PRINT " 1200" ..
ENDIF ..
IF baud=8 ..
    PRINT " 600" ..
ENDIF ..
IF baud=9 ..
    PRINT " 300" ..
ENDIF ..
IF baud=10 ..
    PRINT " 200" ..
ENDIF ..
IF baud=11 ..
    PRINT " 150" ..
ENDIF ..
IF baud=12 ..
    PRINT " 134" ..
ENDIF ..
IF baud=13 ..
    PRINT " 110" ..
ENDIF ..
IF baud=14 ..
    PRINT " 75" ..
ENDIF ..
IF baud=15 ..
    PRINT " 50" ..
ENDIF ..
PRINT "Printer: "; ..
IF BIOS(&H8,0)=1 ..
    IF prnt! ..
        PRINT "ON      " ..
    ELSE ..
        PRINT "OFF      " ..
    ENDIF ..
ELSE ..
    PRINT "not available" ..
```

```
ENDIF ↴
PRINT "Capture Buffer: "; ↴
IF capture!=-1 ↴
  PRINT "OPEN " ↴
ENDIF ↴
IF capture!=0 ↴
  PRINT "CLOSED" ↴
ENDIF ↴
PRINT "Capture "; ↴
IF LEN(capture$)=0 ↴
  PRINT "EMPTY" ↴
ELSE ↴
  size=INT((LEN(capture$)/32767)*100) ↴
  IF size=0 ↴
    PRINT CHR$(27);";p <1%" ↴
  ELSE ↴
    PRINT size;"%" ↴
  ENDIF ↴
ENDIF ↴
PRINT "Disk Verify "; ↴
IF PEEK(&H444)=1 ↴
  PRINT "OFF" ↴
ELSE ↴
  PRINT "ON " ↴
ENDIF ↴
drive=GEMDOS(&H19) ↴
PRINT "Active Disk Drive is: ";CHR$(65+drive) ↴
IF path_name$<>"" ↴
  PRINT "Directory: ";path_name$ ↴
ENDIF ↴
PRINT DATE$;" ";TIME$ ↴
' ↴
IF rez=1 ↴
  PRINT CHR$(27);";b";CHR$(3); ↴
ENDIF ↴
RETURN ↴
```

This next procedure simply removes check marks from the menu items.

```
PROCEDURE delete_check ..  
' ..  
FOR ctr%=65 TO 70 ..  
    MENU ctr%,0      !not checked ..  
NEXT ctr% ..  
' ..  
RETURN ..  
' ..  
PROCEDURE remove_check ..  
FOR ctr%=11 TO 26 ..  
    MENU ctr%,0 ..  
NEXT ctr% ..  
RETURN ..  
' ..
```

Now come the uploading and downloading routines. ASCII is very simple, just send the characters.

Xmodem is rather confusing. For the more curious, an explanation of xmodem follows this program.

```
' ..  
PROCEDURE dl_ascii ..  
' ..  
FILESELECT CHR$(drive+65)+":\*.*",b$,filename$ ..  
' ..  
IF filename$="" ..  
    GOTO skip_ascii_dl ..  
ENDIF ..  
@have_file ..  
IF is_it=0 ..  
    alrt$="File Exists!" ..  
    ALERT 1,alrt$,2,"Replace|Cancel",b ..  
    IF b=1 ..  
        filename$="" ..  
        GOTO skip_ascii_dl ..  
    ENDIF ..  
ENDIF ..
```

```
'  
PRINT "Press <Esc> to exit" ↴  
OPEN "O",#3,filename$ ↴  
' ↴  
OUT 1,xon ↴  
' ↴  
REPEAT ↴  
' ↴  
IF INP?(1) ↴  
byte=INP(1) ↴  
OUT #3,byte ↴  
OUT 2,byte ↴  
ENDIF ↴  
' ↴  
UNTIL ASC(INKEY$)=27 ↴  
' ↴  
CLOSE #3 ↴  
' ↴  
skip_ascii_dl: ↴  
' ↴  
@rt_btn ↴  
' ↴  
RETURN ↴  
  
PROCEDURE ul_ascii ↴  
' ↴  
FILESELECT CHR$(drive+65)+":\*.*",b$,filename$ ↴  
' ↴  
IF filename$="" ↴  
GOTO skip_ascii_ul ↴  
ENDIF ↴  
@have_file ↴  
IF is_it=-33 ↴  
alrt$="File doesn't exist!" ↴  
ALERT 1,alrt$,1,"OK",b ↴  
filename$="" ↴  
ENDIF ↴  
' ↴  
OPEN "I",#3,filename$ ↴  
' ↴  
OUT 1,xon ↴
```

```
' ↴
DO ↴
' ↴
EXIT IF EOF(#3) ↴
byte=INP(#3) ↴
OUT 1,byte ↴
OUT 2,byte ↴
PAUSE 5 ↴
' ↴
LOOP ↴
' ↴
CLOSE #3 ↴
' ↴
skip_ascii_ul: ↴
'
RETURN ↴
```

Here is Xmodem_receive.

```
PROCEDURE xmodem_receive ↴
' ↴
xcol=CRSCOL ↴
ylin=CRSLIN ↴
SGET screen$ ↴
FILESELECT CHR$(drive+65)+":\*.*",b$,filename$ ↴
IF filename$="" ↴
GOTO skip_xmodem_rcv ↴
ENDIF ↴
@have_file ↴
IF is_it=0 ↴
alrt$="File Exists!" ↴
ALERT 1,alrt$,2,"Replace|Cancel",b ↴
IF b=2 ↴
filename$="" ↴
GOTO skip_xmodem_rcv ↴
ENDIF ↴
ENDIF ↴
@do_rcv ↴
skip_xmodem_rcv: ↴
```

```
SPUT screen$ ..  
PRINT AT(xcol,ylin); ..  
RETURN ..  
  
PROCEDURE do_rcv ..  
tucr=ucr ..  
ucr=&X10010000      !8 bits, no parity, 1 stop ..  
@rsconf ..  
VOID XBIOS(21,0) ..  
CLS ..  
y1=180 ..  
y2=185 ..  
IF rez=2 ..  
    y1=y1*2 ..  
    y2=y2*2 ..  
ENDIF ..  
BOX 45,50,595,y1 ..  
BOX 40,45,600,y2 ..  
' ..  
rcv_start: ..  
' ..  
OPEN "O",#3,filename$ ..  
PRINT AT(27,10); "XMODEM Protocol (Receive)" ..  
place=(80-(LEN(filename$)+12))/2 ..  
PRINT AT(place,13); "Receiving: ";UPPER$(filename$) ..  
block%=1 ..  
attempt%=1 ..  
' ..  
send_nak: ..  
' ..  
@clear_modem ..  
VOID BIOS(3,1,nak) ..  
time_out=0 ..  
' ..  
status_line: ..  
' ..  
PRINT AT(26,17); "Block--> ";block% ..  
PRINT AT(46,17); "Attempt--> ";attempt% ..  
IF have!=TRUE ..  
    have!=FALSE ..  
    OUT 1,ack ..
```

```
ENDIF ↴
'
rcv_bconstat: ↴
' ↴
stat!=BIOS(1,1) ↴
IF stat!=0 THEN ↴
    GOTO rcv_time_out ↴
ELSE ↴
    GOTO rcv_byte ↴
ENDIF ↴
' ↴
rcv_time_out: ↴
' ↴
INC time_out ↴
IF (time_out/128)=(128*10) THEN ↴
    GOTO blew_it ↴
ENDIF ↴
IF (time_out/128)=INT(time_out/128) THEN ↴
    GOTO send_nak ↴
ELSE ↴
    GOTO rcv_bconstat ↴
ENDIF ↴
' ↴
rcv_byte: ↴
' ↴
da%=BIOS(2,1) ↴
IF da%=soh THEN ↴
    GOTO rcv_block_count ↴
ENDIF ↴
IF da%=eot THEN ↴
    GOTO rcvd_it ↴
ENDIF ↴
IF da%=can THEN ↴
    GOTO blew_it ↴
ENDIF ↴
GOTO rcv_time_out ↴
'
rcv_block_count: ↴
' ↴
block numb%=BIOS(2,1) ↴
block comp%=BIOS(2,1) ↴
```

```
check%=0 ..  
block$="" ..  
checksum%=0 ..  
FOR byte=1 TO 128 ..  
    da%=0 ..  
    byte_data: ..  
    abort$=INKEY$ ..  
    EXIT IF abort$=CHR$(24) ..  
    IF BIOS(1,1) THEN ..  
        da%=BIOS(2,1) ..  
        check%=check%+da% ..  
        block$=block$+CHR$(da%) ..  
    ELSE ..  
        GOTO byte_data ..  
    ENDIF ..  
NEXT byte ..  
IF abort$=CHR$(24) THEN ..  
    GOTO blew_it ..  
ELSE ..  
    checksum%=check% AND 255 ..  
ENDIF ..  
' ..  
check: ..  
' ..  
check_sum%=0 ..  
IF BIOS(1,1)=-1 THEN ..  
    check_sum%=INP(1) ..  
ELSE ..  
    GOTO check ..  
ENDIF ..  
IF checksum%=check_sum% THEN ..  
    GOTO build_block ..  
ELSE ..  
    GOTO check_attempts ..  
ENDIF ..  
' ..  
build_block: ..  
' ..  
PRINT #3,block$; ..  
attempt%=1 ..  
INC block% ..
```

```
time_out=0 ↴
b_out=0 ↴
@clear_modem ↴
have!=TRUE ↴
GOTO status_line ↴
' ↴
check_attempts: ↴
' ↴
INC attempt% ↴
IF attempt%>10 THEN ↴
    GOTO blew_it ↴
ELSE ↴
    @clear_modem ↴
ENDIF ↴
GOTO send_nak ↴
' ↴
blew_it: ↴
' ↴
OUT 1,can ↴
OUT 1,can ↴
OUT 1,can ↴
CLOSE #3 ↴
KILL filename$ ↴
PRINT AT(10,20); "Status: XMODEM CANCELLED -- "; ↴
PRINT UPPER$(filename$); " deleted." ↴
GOTO xmodem_end ↴
' ↴
rcvd_it: ↴
' ↴
PAUSE 50 ↴
OUT 1,ack ↴
CLOSE #3 ↴
PRINT AT(10,20); "Status: XMODEM COMPLETE. "; ↴
PRINT UPPER$(filename$); ↴
PRINT " received." ↴
GOTO xmodem_end ↴
' ↴
xmodem_end: ↴
' ↴
flag_it=1 ↴
FOR ctr%=0 TO 2 ↴
```

```
PRINT CHR$(7)      ! Ring bell ↴
PAUSE 50 ↴
NEXT ctr% ↴
ucr=tucr ↴
@rsconf ↴
' ↴
RETURN ↴
' ↴
PROCEDURE clear_modem ↴
' ↴
DO ↴
test!=BIOS(1,1) ↴
EXIT IF test!=0 ↴
VOID INP(1) ↴
LOOP ↴
' ↴
RETURN ↴
' ↴
```

The next procedure changes the displayed baud rate, and also calls XBIOS \$15 to change the baud rate.

```
' ↴
PROCEDURE baud_rate ↴
' ↴
IF MENU(0)=11 ↴
@remove_check ↴
MENU 11,1 ↴
baud=0 ↴
ENDIF ↴
' ↴
IF MENU(0)=12 ↴
@remove_check ↴
MENU 12,1 ↴
baud=1 ↴
ENDIF ↴
' ↴
IF MENU(0)=13 ↴
@remove_check ↴
```

```
MENU 13,1 ↴
baud=2 ↴
ENDIF ↴
' ↴
IF MENU(0)=14 ↴
@remove_check ↴
baud=3 ↴
MENU 14,1 ↴
ENDIF ↴
' ↴
IF MENU(0)=15 ↴
@remove_check ↴
baud=4 ↴
MENU 15,1 ↴
ENDIF ↴
' ↴
IF MENU(0)=16 ↴
@remove_check ↴
baud=5 ↴
MENU 16,1 ↴
ENDIF ↴
' ↴
' ↴
IF MENU(0)=17 ↴
@remove_check ↴
baud=6 ↴
MENU 17,1 ↴
ENDIF ↴
' ↴
IF MENU(0)=18 ↴
@remove_check ↴
baud=7 ↴
MENU 18,1 ↴
ENDIF ↴
' ↴
IF MENU(0)=19 ↴
@remove_check ↴
baud=8 ↴
MENU 19,1 ↴
ENDIF ↴
' ↴
```

```
IF MENU(0)=20 ..  
  @remove_check ..  
  baud=9 ..  
  MENU 20,1 ..  
ENDIF ..  
' ..  
IF MENU(0)=21 ..  
  @remove_check ..  
  baud=10 ..  
  MENU 21,1 ..  
ENDIF ..  
' ..  
IF MENU(0)=22 ..  
  @remove_check ..  
  baud=11 ..  
  MENU 22,1 ..  
ENDIF ..  
' ..  
IF MENU(0)=23 ..  
  @remove_check ..  
  baud=12 ..  
  MENU 23,1 ..  
ENDIF ..  
' ..  
IF MENU(0)=24 ..  
  @remove_check ..  
  baud=13 ..  
  MENU 24,1 ..  
ENDIF ..  
' ..  
IF MENU(0)=25 ..  
  @remove_check ..  
  baud=14 ..  
  MENU 25,1 ..  
ENDIF ..  
' ..  
IF MENU(0)=26 ..  
  @remove_check ..  
  baud=15 ..  
  MENU 26,1 ..  
ENDIF ..
```

```
'  
RETURN.  
'  
PROCEDURE rsconf.  
'  
VOID XBIOS(rsconf,baud,flow,ucr,rsr,tsr,scr).  
@menu_check.  
'  
RETURN.  
'
```

This procedure is used whenever the “Press Right Mouse Button...” message is required.

```
PROCEDURE rt_btn.  
'  
PRINT AT(15,20);--> Press Right Mouse Button go on line ---.  
'  
RETURN.  
'
```

Here's the xmodem_send procedure.

```
PROCEDURE xmodem_send.  
'  
VOID XBIOS(&H15,0).  
SGET screen$.  
FILESELECT CHR$(drive+65)+":*.*",b$,filename$.  
IF filename$="".  
    GOTO skip_xmodem_send.  
ENDIF.  
@have_file.  
IF is_it=-33.  
    alrt$="File doesn't exist!".  
    ALERT 1,alrt$,1,"OK",b.  
    filename$="".  
    GOTO skip_xmodem_send.  
'
```

```
ENDIF ..  
@do_send ..  
skip_xmodem_send: ..  
SPUT screen$ ..  
VOID XBIOS(&H15,2) ..  
OUT 1,xon ..  
RETURN ..  
' ..  
PROCEDURE do_send ..  
ctr=0 ..  
tucr=ucr ..  
ucr=&X10010000      !8 bits, no parity, 1 stop ..  
@rsconf ..  
OPEN "I",#2,filename$ ..  
file_length=LOF(#2) ..  
CLOSE #2 ..  
xblock=INT(file_length/128)+1 ..  
CLS ..  
ptr=VARPTR(buf) ..  
y1=180 ..  
y2=185 ..  
IF rez=2 ..  
    y1=y1*2 ..  
    y2=y2*2 ..  
ENDIF ..  
BOX 45,50,595,y1 ..  
BOX 40,45,600,y2 ..  
PRINT AT(27,10);"XMODEM Protocol (Transmit)" ..  
place=(80-(LEN(filename$)+7))/2 ..  
PRINT AT(place,13);"Sending:      ";UPPER$(filename$) ..  
PRINT AT(28,15);xblock;" Xmodem blocks in file." ..  
' ..  
e_flag=0 ..  
block%=0 ..  
block_no%=0 ..  
VOID BIOS(3,1,17)  !send XON ..  
time_out=TIMER ..  
OPEN "I",#3,filename$ ..  
' ..  
PRINT AT(30,22);"Waiting for response"; ..  
' ..
```

```
wait_nak: ↴
' ↴
IF INT((TIMER-time_out)/100)=100 ↴
  GOTO cancel ↴
ENDIF ↴
' ↴
stat=INP?(1) ↴
IF stat=0 ↴
  GOTO wait_nak ↴
ENDIF ↴
byte%=BIOS(2,1) ↴
IF byte%<21 ↴
  GOTO wait_nak ↴
ENDIF ↴
send: ↴
check%=0 ↴
try%=0 ↴
' ↴
DO ↴
' ↴
INC block% ↴
EXIT IF EOF(#3) ↴
' ↴
IF block%=256 ↴
  block%=0 ↴
  ctr=1 ↴
ENDIF ↴
block%=block% AND 255 ↴
block_comp%=255-block% ↴
check%=0 ↴
ctr%=0 ↴
WHILE ctr%<127 ↴
  POKE ptr+ctr%,INP(#3) ↴
  IF EOF(#3) ↴
    POKE ptr+ctr%,eot ↴
    e_flag=1 ↴
    ctr%=127 ↴
  ENDIF ↴
  INC ctr% ↴
WEND ↴
' ↴
```

```
r_send: ↴
' ↴
IF ctr=1 ↴
  dblock%=256+block% ↴
ELSE ↴
  dblock%=block% ↴
ENDIF ↴
PRINT AT(21,22);"Sending Block: ";dblock%;" Comp: ";block_comp%;" Try: ";try% ↴
VOID BIOS(3,1,1) !send SOH ↴
VOID BIOS(3,1,block%) ↴
VOID BIOS(3,1,block_comp%) ↴
FOR ctr%=0 TO 127 ↴
  VOID BIOS(3,1,buf(ctr%)) ↴
  check%=(check%+buf(ctr%)) AND 255 ↴
NEXT ctr% ↴
VOID BIOS(3,1,check%) ↴
time_out=TIMER ↴
' ↴
wait_ack: ↴
IF INT((TIMER-time_out)/100)=100 ↴
  GOTO cancel ↴
ENDIF ↴
' ↴
stat=INP?(1) ↴
IF stat=0 ↴
  GOTO wait_ack ↴
ENDIF ↴
byte%=BIOS(2,1) ↴
IF byte%=ack ↴
  try%=0 ↴
  GOTO n_loop ↴
ENDIF ↴
IF byte%=nak ↴
  INC try% ↴
  IF try%>10 ↴
    GOTO cancel ↴
  ENDIF ↴
  GOTO r_send ↴
ENDIF ↴
IF byte%>can ↴
  GOTO cancel ↴
```

```
ENDIF ↴
' ↴
n_loop: ↴
IF e_flag=1 ↴
  GOTO dne_it ↴
ENDIF ↴
LOOP ↴
' ↴
cancel: ↴
OUT 1,can ↴
OUT 1,can ↴
OUT 1,can ↴
PRINT ↴
PRINT AT(10,20);SPACE$(12)+"Excessive Errors -- Procedure Aborted!" ↴
PRINT ↴
dne_it: ↴
PRINT AT(10,22);SPACE$(21)+"Transmission complete!"+SPACE$(19) ↴
' ↴
count=0 ↴
send_eot: ↴
IF count=10 ↴
  GOTO enough ↴
ENDIF ↴
VOID BIOS(3,1,eot)      !send End Of Transmission ↴
stat=INP?(1) ↴
IF stat=0 ↴
  GOTO send_eot ↴
ENDIF ↴
byte%=BIOS(2,1) ↴
IF byte%<>ack ↴
  GOTO send_eot ↴
  INC count ↴
ENDIF ↴
'
enough: ↴
CLOSE #3 ↴
FOR ctr%=0 TO 2 ↴
  PRINT CHR$(7);          ! Ring bell ↴
  PAUSE 50 ↴
NEXT ctr% ↴
PAUSE 100 ↴
```

```
@rt_btnn ..  
@clear_modem ..  
VOID XBIOS(21,1) ..  
' ..  
ucr=tucr ..  
@rsconf ..  
' ..  
RETURN ..  
' ..  
PROCEDURE have_file ..  
' ..  
' Returns -33 if file does not exist, otherwise expect a 0 ..  
' ..  
is_it=GEMDOS(&H4E,L:VARPTR(filename$)) ..  
' ..  
RETURN ..  
' ..
```

Procedure *runit* uses the EXEC function to load and run and program from disk. An area of memory 32K larger than the program size is reserved. If there's not enough free memory available, an alert box will be displayed.

```
PROCEDURE runit ..  
' ..  
MENU OFF ..  
SGET screen$ ..  
CLS ..  
PRINT "Select .PRG, .TOS or .TTP application to run:" ..  
FILESELECT CHR$(drive+65)+":*.*",b$,filename$ ..  
' ..  
@have_file ..  
IF is_it=-33 ..  
    alrt$="File doesn't exist!" ..  
    ALERT 1,alrt$,1,"OK",b ..  
    filename$="" ..  
ENDIF ..  
IF filename$="" ..  
    GOTO skip_runit ..  
ENDIF ..
```

```
' ↴
para$="" ↴
IF UPPER$(RIGHT$(filename$,4))=".PRG" ↴
    GOTO rn_it ↴
ENDIF ↴
IF UPPER$(RIGHT$(filename$,4))=".TOS" ↴
    GOTO rn_it ↴
ENDIF ↴
IF UPPER$(RIGHT$(filename$,4))=".TTP" ↴
    CLS ↴
    LINE INPUT "Enter .TTP parameters: ",para$ ↴
    para$=CHR$(LEN(para$))+para$+CHR$(0) ↴
    GOTO rn_it ↴
ENDIF ↴
' ↴
alrt$="| "+filename$+"| is not an executable file." ↴
ALERT 3,alrt$,1,"OK",b ↴
GOTO skip_runit ↴
' ↴
m_it: ↴
' ↴
CLS ↴
OPEN "I",#2,filename$ ↴
file_length=LOF(#2) ↴
CLOSE #2 ↴
file_length=file_length+32767 ↴
' ↴
IF FRE(0)<file_length ↴
    alrt$="Not enough free memory!!Reboot without .ACC
files|and/or RAM Disk." ↴
    ALERT 1,alrt$,1,"OK",b ↴
    GOTO skip_runit ↴
ENDIF ↴
' ↴
IF UPPER$(RIGHT$(filename$,4))=".TOS" ↴
    VOID XBIOS(21,1) ↴
ENDIF ↴
IF UPPER$(RIGHT$(filename$,4))=".TTP" ↴
    VOID XBIOS(21,1) ↴
ENDIF ↴
IF UPPER$(RIGHT$(filename$,4))=".PRG" ↴
```

```
SHOWM ↴
ENDIF ↴
' ↴
RESERVE FRE(0)-file_length ↴
PRINT AT(10,5);"LOADing ";filename$ ↴
EXEC 0,filename$,para$,"" ↴
RESERVE FRE(0)+(file_length-255) ↴
' ↴
skip_runit: ↴
VOID XBIOS(21,0) ↴
SPUT screen$ ↴
' ↴
RETURN ↴
' ↴
```

This procedure takes care of autodialing phone numbers.

```
PROCEDURE auto_dial ↴
' ↴
IF dial_now=1 ↴
  GOTO auto_done ↴
ENDIF ↴
alrt$=SPACE$(8)+"EasyTerm | "+SPACE$(9)+"Auto-Dial"+SPACE$(7)+"| Load new phone list?" ↴
ALERT 2,alrt$,2,"Yes|No",b ↴
IF b=1 ↴
  CLS ↴
  dr_load: ↴
  b$="autodial.phn" ↴
  FILESELECT ↴
  CHR$(drive+65)+":\path_name$+\*.phn",b$,filename$ ↴
  IF filename$="" ↴
    GOTO dr_fini ↴
  ENDIF ↴
  @have_file ↴
  IF is_it=-33 ↴
    alrt$="Can't find that file." ↴
    ALERT 1,alrt$,1,"OK",b ↴
    GOTO dr_load ↴
  ENDIF ↴
```

```
ENDIF ↴
' ↴
' ↴
DEFMOUSE 2 ↴
OPEN "I",#3,filename$ ↴
ctr%=0 ↴
FOR ctr%=0 TO 9 ↴
    LINE INPUT ↴ #3,service$(ctr%);number$(ctr%);pass$(ctr%);prompt$(ctr%) ↴
NEXT ctr% ↴
CLOSE #3 ↴
DEFMOUSE 0 ↴
dr_fini: ↴
b$="" ↴
ENDIF ↴
' ↴
alrt$="      EasyTerm ||      Auto-Dial      " ↴
ALERT 2,alrt$,3,"Edit No.|Dial No.|Cancel",b ↴
IF b=1 ↴
    ed_number: ↴
    CLS ↴
    @display_auto ↴
    INPUT "Edit Number (enter 'A' to Abort ";number$ ↴
    IF UPPER$(number$)="A" ↴
        CLS ↴
        GOTO auto_done ↴
    ENDIF ↴
    ' ↴
    IF VAL(number$)<1 OR VAL(number$)>10 ↴
        PRINT "Invalid number. Select again" ↴
        PAUSE 200 ↴
        GOTO ed_number ↴
    ENDIF ↴
    CLS ↴
    number=VAL(number$)-1 ↴
    ' ↴
    re_do: ↴
    PRINT "Presently: ",service$(number);TAB(30);number$(number) ↴
    LINE INPUT "Change Service name to: ",service$(number) ↴
    LINE INPUT "Change modem command and phone number string to: ";number$(number) ↴
    LINE INPUT "Enter password sequence or press <Return>: ";pass$(number) ↴
    IF pass$(number)<>"" ↴
```

```
LINE INPUT "Enter Prompt from Service for Password:";prompt$(number) ↴
ENDIF ↴
PRINT "Entry ";number$;" corrected to" ↴
PRINT service$(number);";";number$(number) ↴
PRINT "Password is: ";pass$(number) ↴
IF pass$(number)<>"" ↴
    PRINT "Prompt is: ";prompt$(number) ↴
ENDIF ↴
INPUT "Is this correct? (Y or N) ";response$ ↴
IF UPPER$(response$)<>"Y" ↴
    GOTO re_do ↴
ENDIF ↴
PRINT ↴
INPUT "Edit another entry? (Y or N) ";response$ ↴
IF UPPER$(response$)="Y" ↴
    GOTO ed_number ↴
ENDIF ↴
choose_file: ↴
CLS ↴
PRINT "Save file as:" ↴
b$="autodial.phn" ↴
FILESELECT CHR$(drive+65)+":"+path_name$+"\autodial.phn",b$,filename$ ↴
IF filename$="" ↴
    GOTO auto_done ↴
ENDIF ↴
@have_file ↴
IF is_it=0 ↴
    alrt$="File Exists!" ↴
    ALERT 1,alrt$,1,"Replace|Cancel",b ↴
    IF b=2 ↴
        GOTO choose_file ↴
        filename$="" ↴
    ENDIF ↴
ENDIF ↴
' ↴
IF filename$="" ↴
    GOTO auto_done ↴
ENDIF ↴
' ↴
PRINT "Saving file as: ";filename$ ↴
DEFMOUSE 2 ↴
```

```
OPEN "O",#3,filename$ ..  
FOR ctr%=0 TO 9 ..  
    PRINT #3;service$(ctr%) ..  
    PRINT #3;number$(ctr%) ..  
    PRINT #3;pass$(ctr%) ..  
    PRINT #3;prompt$(ctr%) ..  
NEXT ctr% ..  
CLOSE #3 ..  
DEFMOUSE 0 ..  
CLS ..  
ENDIF ..  
' ..  
IF b=2 ..  
    dial_number: ..  
    CLS ..  
    @display_auto ..  
    INPUT "Enter number designator for Service you'd like to call or 'A' to Abort. ";number$ ..  
    IF UPPER$(number$)="A" ..  
        CLS ..  
        GOTO auto_done ..  
    ENDIF ..  
' ..  
    IF ASC(number$)<49 OR ASC(number$)>57 ..  
        PRINT "Invalid number. Select again" ..  
        GOTO dial_number ..  
    ENDIF ..  
    number=VAL(number$)-1 ..  
    dial_now=1 ..  
    CLS ..  
    PRINT AT(1,1) ..  
    @status ..  
    PRINT AT(15,20);"--> Press Right Mouse button to dial";service$(number);"  
ENDIF ..  
' ..  
IF b=3 ..  
    CLS ..  
ENDIF ..  
' ..  
auto_done: ..  
' ..  
RETURN ..
```

```
'  
PROCEDURE display_auto '  
'  
PRINT "Telephone Directory" '  
PRINT "      Service";TAB(28);"Modem command string" '  
FOR ctr% = 0 TO 9 '  
    PRINT ctr%+1;"."; service$(ctr%);TAB(25);number$(ctr%);" "; '  
    IF pass$(ctr%)<>"" '  
        PRINT TAB(50);"*Password*"; '  
    ENDIF '  
    PRINT '  
NEXT ctr% '  
RETURN '  
'  
PROCEDURE dial_it '  
try_again: '  
abort=0 '  
@control_c '  
IF abort=1 '  
    PRINT CHR$(27);"p <<Dialing sequence aborted by User request>> ";CHR$(27);"q" '  
    GOTO fini_dial '  
ENDIF '  
cnnct=0 '  
PRINT "Auto Dialing ";service$(number) '  
FOR ctr% = 1 TO LEN(number$(number)) '  
    OUT 1,ASC(MID$(number$(number),ctr%,1)) AND 127 '  
    PAUSE 5 '  
NEXT ctr% '  
OUT 1,13 '  
mode$="" '  
st_time=TIMER '  
DO '  
    @control_c '  
    IF abort=1 '  
        PRINT CHR$(27);"p <<Dialing sequence aborted by User request>> ";CHR$(27);"q" '  
        GOTO fini_dial '  
    ENDIF '  
    IF INP?(1)<>0 '  
        t=INP(1) '  
        OUT 2,t '  
        mode$=mode$+CHR$(t) '  
    ENDIF  
'
```

```
ENDIF ↴
IF UPPER$(RIGHT$(mode$,4))<>"NECT" AND TIMER-st_time>5000 ↴
  ' Pause 1500 ↴
  OUT 1,13 ↴
  GOTO try_again ↴
ENDIF ↴
IF UPPER$(RIGHT$(mode$,4))="NECT" ↴
  cnt=1 ↴
ENDIF ↴
EXIT IF cnt=1 ↴
LOOP ↴
PRINT ↴
' ↴
OUT 1,13 ↴
' ↴
IF pass$(number)="" ↴
  GOTO fini_dial ↴
ENDIF ↴
' ↴
REPEAT ↴
  IF INP?(1)<>0 ↴
    t=INP(1) ↴
    OUT 2,t AND 127 ↴
    mode$=mode$+CHR$(t AND 127) ↴
  ENDIF ↴
UNTIL RIGHT$(mode$,LEN(prompt$(number)))=prompt$(number) ↴
' ↴
FOR ctr%=1 TO LEN(pass$(number)) ↴
  OUT 1,ASC(MID$(pass$(number),ctr%,1)) ↴
  PRINT "*"; ↴
  PAUSE 5 ↴
NEXT ctr% ↴
OUT 1,13 ↴
OUT 1,10 ↴
fini_dial: ↴
dial_now=0 ↴
RETURN ↴ ↴
' ↴
```

This procedure allows the function key definitions to be edited.

```
PROCEDURE edit_function_key ..  
  '..  
  n_key: ..  
  CLS ..  
  PRINT "Function Keys:" ..  
  FOR ctr% = 0 TO 9 ..  
    PRINT "F";ctr%+1;" ";key$(ctr%) ..  
  NEXT ctr% ..  
  PRINT ..  
  PRINT "Press Function key to edit or 'A' to abort:" ..  
  check_key: ..  
  a=INP(2) ..  
  IF a=&H41 OR a=&H61 ..  
    GOTO no_define ..  
  ENDIF ..  
  IF a<187 OR a>196      !check function key ..  
    GOTO check_key ..  
  ENDIF ..  
  a=a-187 ..  
  w_key: ..  
  CLS ..  
  PRINT "Key now defined as:" ..  
  PRINT key$(a) ..  
  PRINT ..  
  PRINT "Enter new definition:" ..  
  LINE INPUT key$(a) ..  
  PRINT ..  
  PRINT "F";a+1;" Defined as:" ..  
  PRINT key$(a) ..  
  PRINT ..  
  INPUT "Is this correct? (Y or N)",response$ ..  
  IF UPPER$(response$)<>"Y" ..  
    GOTO w_key ..  
  ENDIF ..  
  PRINT ..  
  INPUT "Edit another entry? (Y or N)",response$ ..  
  IF UPPER$(response$)="Y" ..  
    GOTO n_key ..  
  ENDIF ..
```

```
' ↴
CLS ↴
alrt$=" |Save redefined Function Keys?" ↴
ALERT 2,alrt$,1,"YES|NO",b ↴
' ↴
IF b=1 ↴
  CLS ↴
  PRINT "Saving Function Keys....." ↴
  ' ↴
  DEFMOUSE 2 ↴
  OPEN "O",#3,"funct.key" ↴
  FOR ctr%=0 TO 19 ↴
    PRINT #3;key$(ctr%) ↴
  NEXT ctr% ↴
  CLOSE #3 ↴
  DEFMOUSE 0 ↴
ENDIF ↴
' ↴
no_define: ↴
CLS ↴
' ↴
RETURN ↴
' ↴
PROCEDURE edit_shiftfunction_key ↴
' ↴
sn_key: ↴
CLS ↴
PRINT "Function Keys:" ↴
FOR ctr%=0 TO 9 ↴
  PRINT "Shifted F";ctr%+1;" ";key$(ctr%) ↴
NEXT ctr% ↴
PRINT ↴
PRINT "Press Function key to edit or 'A' to abort:" ↴
scheck_key: ↴
a=INP(2) ↴
IF a=&H41 OR a=&H61 ↴
  GOTO no_sdefine ↴
ENDIF ↴
IF a<212 OR a>221      !check function key ↴
  GOTO scheck_key ↴
ENDIF ↴
```

```
a=a-202 ..  
s_key: ..  
CLS ..  
PRINT "Key now defined as:" ..  
PRINT key$(a) ..  
PRINT ..  
PRINT "Enter new definition:" ..  
LINE INPUT key$(a) ..  
PRINT ..  
PRINT "Shifted F";a+1;" Defined as:" ..  
PRINT key$(a) ..  
PRINT ..  
INPUT "Is this correct? (Y or N) ",response$ ..  
IF UPPER$(response$)<>"Y" ..  
    GOTO s_key ..  
ENDIF ..  
PRINT ..  
INPUT "Edit another entry? (Y or N)",response$ ..  
IF UPPER$(response$)="Y" ..  
    GOTO sn_key ..  
ENDIF ..  
' ..  
alrt$=" |Save redefined Function Keys?" ..  
ALERT 2,alrt$,1,"YES|NO",b ..  
' ..  
IF b=1 ..  
    CLS ..  
    PRINT "Saving Function Keys....." ..  
    ' ..  
    DEFMOUSE 2 ..  
    OPEN "O",#3,"funct.key" ..  
    FOR ctr%=0 TO 19 ..  
        PRINT #3;key$(ctr%) ..  
    NEXT ctr% ..  
    CLOSE #3 ..  
    DEFMOUSE 0 ..  
ENDIF ..  
' ..  
no_sdefine: ..  
CLS ..  
' ..
```

```
RETURN ↴
' ↴
```

This procedure uses the defined function keys.

```
PROCEDURE function_key ↴
' ↴
c=t-187 ↴
FOR ctr%=1 TO LEN(key$(c)) ↴
  IF ASC(MID$(key$(c),ctr%,1))=&H7E ↴
    OUT 2,13 ↴
    OUT 1,13 ↴
    GOTO nw_key ↴
  ENDIF ↴
  OUT 1,ASC(MID$(key$(c),ctr%,1)) ↴
  OUT 2,ASC(MID$(key$(c),ctr%,1)) ↴
nw_key: ↴
  PAUSE 1 ↴
NEXT ctr% ↴
t=0 ↴
' ↴
RETURN ↴
' ↴
PROCEDURE sfunction_key ↴
' ↴
c=t-201 ↴
FOR ctr%=1 TO LEN(key$(c)) ↴
  IF ASC(MID$(key$(c),ctr%,1))=&H7E ↴
    OUT 2,13 ↴
    OUT 1,13 ↴
    GOTO snw_key ↴
  ENDIF ↴
  OUT 1,ASC(MID$(key$(c),ctr%,1)) ↴
  OUT 2,ASC(MID$(key$(c),ctr%,1)) ↴
snw_key: ↴
  PAUSE 1 ↴
NEXT ctr% ↴
t=0 ↴
' ↴
```

RETURN ↴

Here's the help screen. Some information about every function is displayed.

```
PROCEDURE help ↴
' ↴
x%=CRSCOL ↴
y%=CRSLIN ↴
VOID XBIOS(21,0) ↴
SGET screen$ ↴
CLS ↴
PRINT ↴
PRINT "ALT B - Toggle Capture Buffer (Open/Closed)" ↴
PRINT "ALT C - Clear Capture Buffer" ↴
PRINT "ALT D - Download File" ↴
PRINT "ALT F - Display Free Space on current disk drive" ↴
PRINT "ALT M - Change Duplex Mode (Full/Half)" ↴
PRINT "ALT P - Toggle Printer On/OFF (if printer active)" ↴
PRINT "ALT S - Change Baud rate (2400,1200,300)" ↴
PRINT "ALT U - Upload File" ↴
PRINT "ALT V - Disk Verify toggle (On/OFF)" ↴
PRINT "ALT X - Display EasyTerm Status" ↴
PRINT "ALT Z - Turn on type ahead buffer" ↴
PRINT "CLR - Clear on line screen" ↴
PRINT "Help - Display this screen" ↴
PRINT "Left Mouse Button displays alternate screen." ↴
PRINT ↴
PRINT ↴
PRINT "Press Return to continue..." ↴
REPEAT ↴
UNTIL INP(2)=13 ↴
CLS ↴
PRINT ↴
PRINT "      Script File Commands" ↴
PRINT ↴
PRINT "Command          Function" ↴
PRINT " AD(name)    Dial service from Autodial file." ↴
PRINT " AL          Alarm (Ring bell)" ↴
PRINT " CA          Toggle Capture buffer On/Off" ↴
```

```
PRINT " DE (time) Delay until (time) hh:mm:ss" ↴
PRINT " DX (filename) Download Xmodem file." ↴
PRINT " EC (text) Echo to screen"
PRINT " PR      Toggle printer On/Off" ↴
PRINT " RS (filename) Load RS-232 .CFG file." ↴
PRINT " SE (text) Send characters to RS-232 port." ↴
PRINT " SC (filename) Save capture buffer as filename." ↴
PRINT " UA (filename) Upload an ASCII file." ↴
PRINT " UX (filename) Upload an Xmodem file." ↴
PRINT " WA      Wait for prompt." ↴
PRINT ↴
PRINT ↴
PRINT "Press Return to continue..." ↴
REPEAT ↴
UNTIL INP(2)=13 ↴
CLS ↴
SPUT screen$ ↴
PRINT AT(x%,y%) ↴
t=13 ↴
RETURN ↴
```

Here are two familiar routines.

```
' ↴
' Save Original Color Palette ↴

PROCEDURE save_palette ↴
LOCAL i ↴
FOR ctr%=0 TO 15 ↴
  palette(ctr%)=XBIOS(7,W:ctr%,W:-1) ↴
NEXT ctr% ↴
RETURN ↴
' ↴
' Restore Original Color Palette ↴

PROCEDURE restore_palette ↴
LOCAL ctr% ↴
FOR ctr%=0 TO 15 ↴
  SETCOLOR ctr%,palette(ctr%) ↴
NEXT ctr% ↴
```

```
RETURN ↴
' ↴
```

Next is the procedure which actually parses prepared script files. That is, it searches the strings in the file for recognizable commands, and acts accordingly.

```
PROCEDURE script ↴
CLS ↴
PRINT "Select Script File to execute." ↴
filename$="" ↴
FILESELECT CHR$(drive+65)+":"+path_name$+"\*.FIL",b$,filename$ ↴
DEFMOUSE 2 ↴
@have_file ↴
IF is_it=-33 ↴
  DEFMOUSE 0 ↴
  alrt$=" EasyTerm| |Can't find that file." ↴
  ALERT 1,alrt$,1,"OK",b ↴
  GOTO skip_script ↴
ENDIF ↴
' ↴
CLS ↴
IF filename$="" ↴
  DEFMOUSE 0 ↴
  GOTO skip_script ↴
ENDIF ↴
' ↴
DEFMOUSE 2 ↴
OPEN "I",#3,filename$ ↴
li=0 ↴
DO ↴
  EXIT IF EOF(#3) ↴
  LINE INPUT #3,script$(li) ↴
  INC li ↴
LOOP ↴
CLOSE #3 ↴
DEFMOUSE 0 ↴
' ↴
abort=0 ↴
PRINT "Executing ",filename$ ↴
PRINT "Script file of ",li-1," lines in memory" ↴
```

```
'  
FOR l_no=0 TO li-1 ..  
@control_c ..  
size=LEN(script$(l_no))-3 ..  
'  
IF LEFT$(script$(l_no),2)="EC"      !ECHO to screen ..  
    PRINT RIGHT$(script$(l_no),size) ..  
ENDIF ..  
'  
IF LEFT$(script$(l_no),2)="WA"      !WAI for prompt ..  
    start_timer=TIMER ..  
    wait$=SPACE$(size) ..  
    wait: ..  
    command$=RIGHT$(script$(l_no),(size)) ..  
    IF TIMER-start_timer>15000 ..  
        PRINT "TIME OUT -Script File aborted" ..  
        l_no=li-1 ..  
        GOTO timed_out ..  
    ENDIF ..  
    IF BIOS(&H1,1)<>0 ..  
        a=INP(1) ..  
    ELSE ..  
        GOTO wait ..  
    ENDIF ..  
    OUT 2,a AND 127 ..  
    IF capture! ..  
        @capture_it ..  
    ENDIF ..  
    IF prnt!=-1 ..  
        OUT 0,a ..  
        IF t=13 ..  
            INC prnt_cnt ..  
            IF prnt_cnt=80 ..  
                OUT 0,13 ..  
                OUT 0,10 ..  
                prnt_cnt=0 ..  
            ENDIF ..  
        ENDIF ..  
    ENDIF ..  
    wait$=wait$+CHR$(a AND 127) ..  
    wait$=RIGHT$(wait$,5) ..
```

```
IF RIGHTS(wait$,size)<>command$ ..  
    GOTO wait ..  
ENDIF ..  
ENDIF ..  
' ..  
IF LEFT$(script$(l_no),2)="SE"      !SEnd output to modem port ..  
IF prnt!=-1 ..  
    LPRINT RIGHT$(script$(l_no),(size)) ..  
ENDIF ..  
FOR s=4 TO LEN(script$(l_no)) ..  
    OUT 1,ASC(MID$(script$(l_no),s,1)) ..  
    OUT 2,ASC(MID$(script$(l_no),s,1)) ..  
NEXT s ..  
OUT 1,13 ..  
OUT 2,13 ..  
OUT 2,10 ..  
ENDIF ..  
' ..  
IF LEFT$(script$(l_no),2)="DE"      !Wait for time delay ..  
time_reached=0 ..  
PRINT "Start time: ";TIME$ ..  
hold_time$=RIGHT$(script$(l_no),size) ..  
w_hour=VAL(LEFT$(hold_time$,2)) ..  
w_min=VAL(MID$(hold_time$,3,2)) ..  
PRINT "Waiting until: ";w_hour;":"; !Hold_time$ ..  
IF w_min<=9 ..  
    PRINT "0";w_min ..  
ELSE ..  
    PRINT w_min ..  
ENDIF ..  
xcol=CRSCOL ..  
yrow=CRSLIN ..  
DO ..  
    abort=0 ..  
    @control_c ..  
    PRINT AT(xcol,yrow);"Time now is: ";TIME$ ..  
    p_time$=LEFT$(TIME$,5) ..  
    IF w_hour=VAL(LEFT$(p_time$,2)) ..  
        IF w_min=VAL(RIGHT$(p_time$,2)) ..  
            time_reached=1 ..  
        ENDIF ..
```

```
ENDIF ↴
EXIT IF time_reached=1 ↴
EXIT IF abort=1 ↴
LOOP ↴
ENDIF ↴
' ↴
IF LEFT$(script$(l_no),2)="AL" ↴
PAUSE 50 ↴
PRINT CHR$(7) ↴
ENDIF ↴
' ↴
IF LEFT$(script$(l_no),2)="PR" ↴
IF prnt!=0 AND prnt=-1 ↴
prnt!=-1 ↴
PRINT ↴
PRINT CHR$(27);"

<PRINTER ON>";CHR$(27);"" ↴
GOTO pr_changed ↴
ENDIF ↴
IF prnt!=-1 AND prnt=-1 ↴
prnt!=0 ↴
PRINT ↴
PRINT CHR$(27);"

<PRINTER OFF>";CHR$(27);"" ↴
ENDIF ↴
pr_changed: ↴
ENDIF ↴
' ↴
IF LEFT$(script$(l_no),2)="CA" ↴
IF capture!=-1 ↴
capture!=0 ↴
PRINT ↴
PRINT CHR$(27);"

<CAPTURE BUFFER CLOSED>";CHR$(27);"" ↴
GOTO changed_cap ↴
ENDIF ↴
IF capture!=0 ↴
capture!=-1 ↴
capture=0 ↴
PRINT ↴
PRINT CHR$(27);"

<CAPTURE BUFFER OPEN>";CHR$(27);"" ↴
ENDIF ↴
changed_cap: ↴
ENDIF ↴


```

```
'  
IF LEFT$(script$(l_no),2)="SC" ..  
filename$=RIGHT$(script$(l_no),size) ..  
DEFMOUSE 2 ..  
OPEN "O",#3,filename$ ..  
PRINT #3,capture$ ..  
CLOSE #3 ..  
DEFMOUSE 0 ..  
ENDIF ..  
'  
IF LEFT$(script$(l_no),2)="DA" ..  
OUT 1,xoff ..  
filename$=RIGHT$(script$(l_no),size) ..  
OPEN "O",#3,filename$ ..  
'  
OUT 1,xon ..  
'  
REPEAT ..  
'  
IF INP?(1) ..  
byte=INP(1) ..  
OUT #3,byte ..  
OUT 2,byte ..  
ENDIF ..  
'  
UNTIL ASC(INKEY$)=27 ..  
'  
CLOSE #3 ..  
ENDIF ..  
'  
IF LEFT$(script$(l_no),2)="DX"      !Xmodem download ..  
OUT 1,xoff ..  
filename$=RIGHT$(script$(l_no),size) ..  
OUT 1,xon ..  
@do_rcv ..  
ENDIF ..  
'  
IF LEFT$(script$(l_no),2)="UA" ..  
OUT 1,xoff ..  
filename$=RIGHT$(script$(l_no),size) ..  
OPEN "I",#3,filename$ ..
```

```
' ↴
OUT 1,xon ↴
' ↴
DO ↴
    EXIT IF EOF(#3) ↴
    byte=INP(#3) ↴
    OUT 1,byte ↴
    OUT 2,byte ↴
    PAUSE 5 ↴
LOOP ↴
' ↴
CLOSE #3 ↴
ENDIF ↴
' ↴
IF LEFT$(script$(l_no),2)="UX" ↴
    OUT 1,xoff ↴
    filename$=RIGHT$(script$(l_no),size) ↴
    @do_send ↴
ENDIF ↴
' ↴
IF LEFT$(script$(l_no),2)="AD" ↴
    number=99 ↴
    nme$=RIGHT$(script$(l_no),size) ↴
    FOR ctr% =0 TO 9 ↴
        IF nme$=service$(ctr%) ↴
            number=ctr% ↴
            ctr%=9 ↴
        ENDIF ↴
    NEXT ctr% ↴
    IF ctr%=9 AND number=99 ↴
        PRINT CHR$(27);"

<FATAL ERROR - Process aborted>";CHR$(27);"" ↴
        l_no=li ↴
        GOTO timed_out ↴
    ENDIF ↴
    @dial_it ↴
ENDIF ↴
' ↴
' ↴
IF LEFT$(script$(l_no),2)="RS" ↴
    filename$=RIGHT$(script$(l_no),size) ↴
    report=0 ↴


```

```
@load_cfg ..  
IF report=0 ..  
    PRINT CHR$(27);"

<FATAL ERROR - Process aborted>" ..  
";CHR$(27);"q" ..  
    l_no=li ..  
    GOTO timed_out ..  
ENDIF ..  
ENDIF ..  
' ..  
timed_out: ..  
IF abort=1 ..  
    l_no=li ..  
    PRINT "Procedure aborted by User" ..  
    OUT 1,ASC["+"] ..  
    OUT 1,ASC["+"] ..  
    OUT 1,ASC["+"] ..  
    OUT 1,13 ..  
    PAUSE 50 ..  
    OUT 1,ASC("A") ..  
    OUT 1,ASC("T") ..  
    OUT 1,ASC("H") ..  
    OUT 1,ASC("O") ..  
ENDIF ..  
NEXT l_no ..  
' ..  
script_fini: ..  
' ..  
PRINT ..  
PRINT filename$;" complete" ..  
PRINT ..  
PRINT "      --->> Press any key to continue <<---" ..  
a=INP(2) ..  
skip_script: ..  
CLS ..  
' ..  
RETURN ..  
' ..  
PROCEDURE control_c ..  
' ..  
IF INP?(2)<>0          !Control C pressed? ..  
    k=INP(2) ..


```

```
IF k=3 ↴
    abort=1 ↴
    l_no=li-1 ↴
ENDIF ↴
ENDIF ↴
' ↴
RETURN ↴
'

PROCEDURE load_cfg ↴
' ↴
PRINT "Select Configuration File to load:" ↴
FILESELECT CHR$(drive+65)+":"+path_name$+"*cfg",b$,filename$ ↴
DEFMOUSE 2 ↴
@have_file ↴
IF is_it=-33 ↴
    DEFMOUSE 0 ↴
    alrt$=" EasyTerm| Can't find that file." ↴
    ALERT 1,alrt$,1,"OK",b ↴
    GOTO rs232_fini ↴
ENDIF ↴
' ↴
CLS ↴
IF filename$="" ↴
    GOTO rs232_fini ↴
ENDIF ↴
' ↴
@do_cfg ↴
report=1 ↴
rs232_fini: ↴
CLS ↴
RETURN ↴
'

PROCEDURE do_cfg ↴
DEFMOUSE 2 ↴
' ↴
OPEN "I",#3,filename$ ↴
' ↴
INPUT #3;flow ↴
INPUT #3;ucr ↴
INPUT #3;rsr ↴
INPUT #3;tsr ↴
```

```
INPUT #3;scr ..  
INPUT #3;duplex ..  
INPUT #3;baud ..  
' ..  
CLOSE #3 ..  
DEFMOUSE 0 ..  
@rsconf ..  
' ..  
RETURN ..
```

Using EasyTerm

EasyTerm, written completely in GFA BASIC, is a powerful yet easy to use tele-communications program for the novice or experienced telecommunicator. EasyTerm offers features missing in many commercial telecommunications programs, including automatic operation from user prepared script files.

If you're new to telecommunications, EasyTerm will have you on line and communicating in no time. While not intended to be everything for everybody, you'll find EasyTerm very useful for most telecommunications purposes. With the Xmodem protocol, up and downloads of files are a breeze. Also, a functional menu permits easy access to DOS commands.

Function selection through menus takes the guess work out of command entry. Just point and click for most options, it's not necessary to remember a list of strange key presses to execute simple functions. However, experts will also find the pre-defined keyboard macros included with EasyTerm handy. A list of these pre-defined keys is included in this chapter, and handy on line Help Screens display can be accessed from the on line screen by pressing the Help key, or by selecting the Help option from the function menu.

EasyTerm utilizes two screens, a white with black text function screen with a menu bar, and a status display, and the bare-bones on line screen, black background with white text, for communicating with other computers. Other text colors may be encountered. When in Half duplex mode, a color monitor will display any text you type in red. Any information passed from within EasyTerm, such as confirmation of turning on the printer with a keyboard macro (ALT-P) from the on line screen will appear in blue. Of course, a monochrome monitor will only display white text on a dark background. From the function screen, press the right mouse button to switch to the on line screen and communicate with your modem or another computer. To transfer from the on line screen to the function screen, press the left mouse button.

Status Display

The status display, in the upper left corner of the function screen, and inserted at the cursor position of the on line screen by pressing <ALT-X>, keeps you up to date on vital communications parameters; Duplex Mode, Baud rate, printer status, buffer status, as well as the time of day.

The printer status will indicate whether your printer is available for output.

The Buffer status line will inform you of the state of the 32K capture buffer. Is the buffer open or closed? What percentage of the total buffer has been used?

The final line of the Status display shows the time and date this line is updated every time the status display is updated, usually updates occur after menu operations.

A Guided Tour

EasyTerm makes communications easy. The preceding information is really all you need to know to go on line. Follow the simple instructions in the next few paragraphs to go on line with GEnie, the General Electric Network for Information Exchange. You'll find a wealth of information in the Atari ST SIG on GEnie, as well as product support from leading software developers for the ST, such as MICHTRON.

Load and run EASYTERM.BAS from within the GFA BASIC Editor/Interpreter. EasyTerm starts by displaying the title screen. Continue by selecting OK with the mouse pointer.

As EasyTerm continues to load, it will check for predefined RS-232 settings in a file with the suffix .CFG. If a file is found, you will be asked if you want to install a configuration file. If a file is selected, those settings will be installed, at the option of the user, rather than the default settings from within EasyTerm. EasyTerm's default configuration settings are Half duplex, 1200 baud, 8 bits, no parity; these settings are used by GEnie. Later, you can create unique files for each service you frequent.

Next, a check is made for any files created with the Auto Dial option. If files are present, the user has the option of loading a phone directory file. This directory stores the modem commands and phone numbers necessary to dial BBS's.

Finally, EasyTerm checks your disk for predefined function keys, again the user has the option of loading these predefined keys, or ignoring them. Once EasyTerm is loaded, and the user definable options are installed, the function screen is displayed. This is your "Home Base" for EasyTerm. Every feature can be accessed easily from the menu options found here.

If you'd like to take advantage of the pre-defined keyboard macros included with EasyTerm, refer to Table 6-2. Any messages from EasyTerm will be displayed in Blue letters (on a Color System).

EasyTerm Functions

EasyTerm offers full control of the RS-232 parameters, allowing configuration for connecting to any host computer system. The selections under Baud on the function screen menu permit setting every baud rate from 19200 to 50, as supported by the ST, from a drop down menu, although baud rates higher than 2400 are, at present, usually useful only for null modem connections to other computers.

Modems supporting high baud rates cost much more than the normal 1200 baud modems which are priced at about \$100 today. Modems supporting up to 9600 baud are available at premium prices. Such high speeds are not supported by any of the major information services at this time.

Slower baud rates (under 300 baud) have been included for use over marginal phone lines and especially for Amateur Radio Operators, experimenting with Packet Radio. Packet Radio is a relatively new process for computer communication via radio. Slow baud rates are often necessary for reliable radio communications.

The selected baud rate is displayed on the drop down menu with a check mark in front of it, as well as on the terminal status display.

A limited amount of manipulation of the baud rate is available from the on line screen by pressing ALT-S (for speed) to execute the keyboard macro which displays an alert box allowing the user to choose 2400, 1200, or 300 baud, the three most popular baud rates for communication.

The next menu heading on the function screen is RS-232, where all parameters, except baud rate, for the configuring the RS-232 output are controlled. Don't worry if you're not sure what to change, the default settings are correct for GEnie. You may need to change the duplex setting to Full duplex for other information services.

The parameter settings available include setting Full or Half duplex modes, word length, parity settings, and the number of stop bits. Duplex settings may also be toggled from the on line screen by pressing <ALT-M> (mode), to select Full or Half duplex modes of operation. The selected duplex mode is displayed with a checkmark on the drop down menu.

When EasyTerm is configured for Half duplex, all input from your keyboard will appear on the on-line screen in red letters, while information received from the host computer will appear as white letters on a black background.

After setting the desired RS-232 parameters, you can save them into a file for installing when EasyTerm is first booted, or for loading from the function menu, by creating a file with the .CFG suffix.

File transfers are a breeze. EasyTerm offers a selection of either ASCII or Xmodem file transfers from the 'File' drop down menu. Select ASCII for straight byte-for-byte transfers of data between computers. This method is usually suitable for text files. For more important files, were missing data could cause a program to crash, Xmodem file transfers are available. The Xmodem (checksum) method has been chosen for EasyTerm, and is available on every major Information Service, and most BBS programs, allowing for a method of virtual error free data transmission.

Select operation from either the menu on the Function screen or from the on line screen by pressing <ALT-D> for downloading files, or <ALT-U> to upload files. From the on line screen, an alert box offers a selection of transfer protocols.

Also selectable from the File section of the Function Screen menu bar is the 32K byte capture buffer status. The capture buffer may be opened by selecting Buffer OPEN from the menu. Close the buffer by selecting Buffer CLOSED. The contents of the buffer may be saved in ASCII format in a text file by selecting SAVE Buffer. A fileselector allows naming the file, or use the default name, CAPTURE.TXT.

The capture buffer may be cleared by selecting Clear buffer from the function menu.

Opening, closing, and clearing the buffer are also available from the online screen by pressing <ALT-B> to toggle the capture buffer open or closed, and <ALT-C> to clear the capture buffer. The save feature is only available from the function screen.

The contents of the capture buffer may be sent to a connected printer by selecting the Print Buffer option from the menu. This menu will be not be selectable, if the program detects no printer ready to receive data from the program. The function cannot be selected accidentally with no printer attached. Most other programs don't check for an attached printer, causing a long delay until the normal ST device timeout takes affect.

The final function of the File menu is Execute File, a means which allows the user to run a .PRG, .TOS, or .TTP files from within EasyTerm. This function will allow you to use files such as ARC.TTP to unarc a file, without exiting from EasyTerm. This is also handy for running an external text editor, such as MicroEmacs, to edit a file, then return to EasyTerm to transmit that file. (EasyTerm does not include a text editor.)

In order to edit text in the capture buffer, save the file to disk, then use the Exec function to edit the file with any text editor. The file may then be saved and reloaded later into the capture buffer for viewing. To prepare a file for transmission, edit the file using and text editor, then save the file and transmit it from EasyTerm as an ASCII UL when prompted by the host system.

Some program files may cause problems for EasyTerm. A section of memory is reserved for the application to be executed. If a dynamic means of memory allocation is utilized, for instance, the allocated memory for the program increases as data is added, data needed by EasyTerm could be overwritten. 1ST Word is a problem program. If insufficient memory exists for properly loading a program, EasyTerm will alert you of the problem, and not attempt to load and execute the program.

Sometimes programs can be run by rebooting the ST, after removing desk accessories, or removing any RAM disk to free up memory. Usually some experimentation is necessary to find out which programs will and won't run properly from within the EasyTerm shell.

EasyTerm DOS

Many useful DOS functions are available from the function screen of EasyTerm. Under the DOS menu, you may select to display the free space on the active disk, to create a folder on the disk, erase files on the disk, rename a file, or display a directory of files. The active drive may also be changed from the DOS menu.

To select the active drive, point to the desired disk, A-F, on the DOS menu, and click the right mouse button. Unavailable disks are not selected. Even single drive users will find drives A and B available, as the ST assumes that in the minimum configuration, two drives may be accessed by an application. Drives not available are shown in pale type, and may not be selected. If you are using a RAM disk with an identifier higher than F, it will need to be re-installed with a letter, preferably D, E, or F.

Auto Dial

The Utilities menu of EasyTerm offers the user a host of useful functions that no terminal program should be without. The first function under the Utilities menu heading is Auto Dial. Note that this function will only work properly if your modem has the ability to automatically dial a number. When this function is selected, an alert box asks whether a new Auto Dial file should be loaded. If the user selects Yes, a file selector box is displayed, allowing the user to select the phone file to load.

If No is selected, an alert box then questions whether the user wants to edit the file, dial a number or cancel this selection. Select the edit box with the mouse pointer to enter or change an existing file, or to enter new information. The phone file in memory is then displayed. If no numbers have been entered yet, a screen showing 10 possible positions is shown. Each file is limited to ten numbers, but the number of files available is unlimited. Create as many as you need for your use.

At this point, the edit process may be cancelled by entering A, and pressing <Return>. To change or create a new entry selected the entry to change by pressing the proper number key, then enter the data as prompted on screen.

First enter a name for the system so you can easily identify the entry, then enter the necessary information for your Auto Dial modem. Be sure to include the command string for your modem. For a Hayes compatible modem, attached to a phone line with touch tone dialing, this string is ATDT, and must be in capital letters. Follow this string with the desired number, without pressing <Return>. When <Return> is pressed, the entry will be placed in memory.

Next you have the option of placing your password in memory for automatically logging onto a service after dialing. Enter your password exactly as required. Finally, enter the prompt you receive for logging on. On GENie, this prompt is U#=. Last, you'll be asked if you'd like to edit another number. Repeat this process until all the information you need to store is entered, or until 10 entries have been made. If you don't want your password stored on the disk for security purposes, just press <Return>.

After all entries have been made, the file may be saved. Select a filename, ending with .PHN, and the next time you load EasyTerm, you're personal Auto dial directory will be ready to load. To dial a number previously entered, select the Dial box with the mouse pointer. When the selectable options are displayed, enter the number which corresponds with the BBS or information service you'd like to call. To cancel at this point, enter the letter 'A'. EasyTerm will return you to the function menu. When you're ready to dial, press the right mouse button. Dialing is automatic from this point.

Printer Options.

If a printer is attached and ready to receive characters from your ST, the Printer ON and Printer OFF options on the Utilities menu, as well as the Print Buffer option on the Files menu, will be selectable.

A note of caution is warranted here. At higher baud rates (above 1200 baud) the printer may not be able to keep up with the flow of characters, and may actually interfere with the smooth transfer of data. It is advisable to have previously installed a print spooler prior to sending online output to the printer. The indication that the printer output is interfering with the flow from the host system is characters lost during the transfer of data.

An alert box will inform you if you are attempting to direct output to the printer at marginal baud rates. You will be permitted to send characters to the printer. It's up to you to detect any speed problems and take the appropriate action. Some baud rates may be totally unacceptable for this function. During testing speeds up to 1200 baud where acceptable.

Set time and Date.

The next available function from the Utilities menu is for setting the time and date. Follow the on screen prompts to enter the proper date and time. Although not absolutely necessary, the date and time stamp are handy for identifying any files created.

If you have installed a time card, the proper date and time will be used without any user intervention. The date and time may be checked from within EasyTerm by using this function, without changing any time or date values. Just press <Return> at each prompt to retain the pre-set date and time.

Define Function Keys

Twenty function keys, F1-F10, and shift F1-shift F10, are available to the user. To define function keys, select this option, then select which set of keys to define or change. When the key entries are displayed, press the key you'd like to define, then enter the data for the key. If a return character (ASCII 13) is needed, for example as part of a password sequence, use the tilde (~) key to mark the position of the return character. (This key is a shifted key and is located as the second key from the right on the top row of the keyboard, next to the backspace key.)

Use the defined function keys when on line simply by pressing the appropriate function key combination.

Disk Write Verify

If additional speed is desired when writing a file to disk, for example when downloading a file, select the Disk Verify OFF option from the Utilities menu. The operating system will now be modified so that disk writing does not use the verify routine.

To re-enable write verify, select Disk Verify On. The default condition is disk verify on.

Help

On line help is available by pressing the Help key. The same screen maybe displayed from within the Function Screen by selecting the Help option from the menu.

Script Files

The most powerful feature of EasyTerm is its ability to process prepared script files automatically. Using a series of 2 letter commands, EasyTerm can examine a Script file, which can be written using any text editor which can save a file in ASCII format, and conduct unattended operation of your ST. Entering Control-C will interrupt and abort the file. This allows you to automatically call an information service, when you're away from your computer.

Note the script file processing is a mini language processor, and it's up to the user to program the correct sequence of commands, just as in using any computer language. A sample script file has been included on the disk.

Table 1 is a list of the commands implemented for Script files. Each command must be two capital letters, followed by a space.

Table 1 List of Script commands

<u>Command</u>	<u>Function</u>
AD (name)	Dial predefined service
AL	Ring Alarm
CA	Toggle capture buffer open/closed
DA (filename)	Download ASCII file
DX (filename)	Download an Xmodem file
DE (time)	Delay execution until indicated time
EC (text)	Echo text string
PR	Toggle printer on/off
RS (file name)	Install previously saved RS-232 .CFG file
SE (message)	Send characters to RS-232 port
SC (filename)	Save capture buffer to disk.
UA (filename)	Upload ASCII file
UX (filename)	Upload and Xmodem file
WA (prompt)	Wait for prompt

Table 2 Predefined Keyboard Macros

ALT-B	Toggle Capture Buffer
ALT-C	Clear Capture Buffer
ALT-D	Download a file
ALT-S	Change baud rate (2400, 1200, 300)
ALT-F	Display Free space on current disk drive
ALT-M	Change Duplex mode (Full or Half)
ALT-P	Toggle Printer On/Off (if printer active)
ALT-U	Upload a file
ALT-V	Toggle Disk write verify (on/Off)
ALT-X	Display terminal status
ALT-Z	Turn on type ahead buffer
CLR	Clear on line screen
Help	Display this (Help) screen

Left Mouse button displays the alternate screen.

A type ahead buffer is available when using the online screen. Press ALT-Z, then type as needed. Input and Output to the modem is halted until the <Return> key is pressed. This function is handy in CB simulators or for on line real time conferencing. The host computer is instructed to temporarily stop sending characters until you have completed typing your message and pressed <Return>. Then all data is transmitted to you, and your message is sent to the host.

The type ahead buffer is quite large. You should never run out of typing space, but most services will only permit you to send several lines at a time. For instance, the maximum type ahead on GEnie appears to be about 10 lines before an error message is transmitted from GEnie.

Xmodem - Getting Technical

The xmodem protocol for data communications is really rather simple.

Data is sent in 128-byte sequentially numbered blocks, with a single checksum byte appended to the end of each block. As the receiving computer acquires the incoming data, it constructs its own checksum and after receiving a block of data, compares its own checksum result with that of the sending computer.

If the receiving computer matches the checksum of the sending computer, it transmits an ACK (ASCII code 6) ACKNOWLEDGE to the sending computer. The ACK means "Everything's okay here, send some more..."

Notice, in the following example, that the sending computer will transmit an "initial NAK" (ASCII 21) NEGATIVE ACKNOWLEDGE or, "Something may be wrong, please send that again".

Due to the asynchronous nature of the initial "hook-up" between the two computers, the receiving computer will "time-out" looking for data, and send the NAK as the signal for the sending computer to begin transmission.

The sending computer knows that the receiving computer will "time-out", and uses this fact to "get in sync". The sending computer responds to the "initial NAK" with a SOH (ASCII code 01) for START OF HEADING, sends the first block number, sends the 1's complement of the block number, sends 128 bytes of 8 bit data (that's why we can transfer ".COM" files), and finally a checksum, where the checksum is calculated by summing the SOH, the block number, the block number 1's complement, and the 128 bytes of data.

This process continues, with the next 128 bytes. If the block was ACK'ed by the receiving computer, and then the next sequential block number and its 1's complement, etc.

If the block is NAK'ed, the sending computer just re-sends the previous block.

If the sending computer transmits a block, the receiving computer gets it and sends an ACK, but the sender doesn't see it, the sending computer thinks that it has failed and after 10 seconds retransmits the block.

The receiving computer has stored the data in memory or on disk, and the receiving computer is now 1 block AHEAD of the transmitting computer!

This is where the operation of the block numbers becomes important. The receiver detects that this is the last block again, and transmits back an ACK, throws away the block, and (effectively) "catches up". What's more, the integrity of the block number is verified by the receiving computer, because it "sums" the SOH (01 Hex) with the block number plus the 1's complement of the block number, and the result MUST BE zero for a proper transfer (e.g. 01 + 01 + FE hex = 00, on the first block).

Normal completion of data transfers will then conclude with an EOT (END OF TRANSMISSION), ASCII 4, from the sending computer, and a final ACK from the receiving computer.

Unfortunately, if the receiving computer misses the EOT, it will continue to wait for the next block (sending a NAK every 10 seconds, up to 10 times) and eventually time-out.

In some cases, where the phone transmission is of poor quality due to weak signals, line noise, and so forth, the receiving computer (and operator) will be provided with an option to quit, after 10 misses.

Appendix A

Error Messages



GFA BASIC Error Message

<u>Error #</u>	<u>Meaning</u>
0	Division by Zero.
1	Overflow
2	Number not integer (-21473648 — +2147483647)
3	Number not byte (0 — 255)
4	Number not Word (0 — 65535)
5	Square Root only for positive numbers.
6	Logarithm only for number greater than zero.
7	Undefined error.
8	Memory full
9	Function or command not possible.
10	String too long (max. size 32,767 characters)
11	Not GFA BASIC V1.0 program.
12	Program too long, memory full, NEW.
13	Not GFA BASIC program file, too short, NEW.
14	Field dimensioned twice.
15	Field not dimensioned.
16	Field index too large.
17	DIM index too large.
18	Wrong number of indexes.
19	Procedure not found.
20	Label not found.
21	On Open only "I"nput "O"utput "R"andom "A"ppend "U"pdate allowed.
22	File already open.
23	File # wrong.
24	File not opened.
25	Input wrong, not numeric.
26	End of file reached.

27	Too many points for Polyline/Polyfill, max. 128.
28	Field must be one dimensional.
29	Number of points larger than field.
30	Merge not an ASCII File
31	Merge line too long — Aborted.
32	==> Syntax Error — program aborted.
33	Label not defined.
34	Insufficient data
35	Data not numeric
36	Syntax error in DATA unpaired quotes.
37	Disk full
38	Command not possible in direct mode.
39	Program error — GOSUB not possible
40	Clear not possible in For — Next loops and procedures.
41	CONT not possible
42	Too few parameters
43	Expression too complex
44	Function not defined
45	Too many parameters
46	Parameter wrong — must be numeric
47	Parameter wrong — must be string
48	Open "R" — Record length wrong
49	Undefined error
50	Not an "R" file
51	Only one field per Open "R" allowed
52	Fields larger than record length
53	Too many fields (max 9)
54	GET/PUT fields string length wrong
55	GET/PUT record number wrong
56	String has wrong length for SPRITE
90	Error in Local

92 Resume (next) not possible Fatal, For or Local
100 Version # of GFA BASIC

TOS and GFA BASIC Compiler Error Messages

<u>Error #</u>	<u>Meaning</u>
-1	General Error
-2	Drive not ready - time exceeded
-3	Undefined error
-4	CRC error - disk checksum wrong.
-5	Bad request - invalid command
-6	Seek error - track not found
-7	Unknown media - boot sector wrong
-8	Sector not found
-9	No paper
-10	Write fault
-11	Read fault
-12	General error 12
-13	Disk write protected
-14	Disk has been changed
-15	Unknown device
-16	Bad sector (verify)
-17	Insert other disk
-32	Invalid function number
-33	File not found
-34	Path not found
-35	Too many files open
-36	Access not possible
-37	Invalid handle
-39	Memory full
-40	Invalid memory block address

-46	Invalid drive ID
-49	No further files
-64	GEMDOS range error - seek wrong?
-65	GEMDOS internal error
-66	Not binary program file
-67	Memory block error

68000 Exception (Bomb) Error Messages

<u>Code</u>	<u>Bombs</u>	<u>Meaning</u>
102	2 bombs	Bus error — PEEK or POKE possibly wrong.
103	3 bombs	Address error. Odd word address. Possibly in DPOKE, DPPEEK, LPOKE, or LPPEEK.
104	4 bombs	Illegal instruction. Execution of an invalid 68000 Assembler command.
105	5 bombs	Division by zero in 68000 ML.
106	6 bombs	CHK exception
107	7 bombs	TRAPV exception
108	8 bombs	Privilege violation
109	9 bombs	Trace exception

Appendix B

BIOS Functions



Appendix B BIOS Functions

This section contains a brief explanation of each of the BIOS Functions of GEM, as it applies to GFA BASIC. A sample routine for calling each function is included. Not all of the routines are functional as listed. Some functions require preliminary set-up by other routines to work correctly. Many BIOS functions are identical to GFA BASIC functions, and will be unnecessary. Each function is listed by its hexadecimal function number, followed by the accepted name for each routine.

NOTE:

BIOS functions permit access to powerful system level functions of the operating system. It is assumed that you have an understanding of these functions prior to using them in your programs, or are willing to assume the risk for any dire consequences encountered.

At the very least, SAVE your program and insert a junk disk in your disk drive before attempting to run any program which may write data to your disk using a BIOS function.

BIOS Functions

\$00

GETMPB

This function tells TOS to copy a memory parameter block into a 24-byte space. This memory parameter block does not appear to be used by TOS.

VOID BIOS(0,L:*pointer*)

pointer A long word (32-bit) containing a pointer to an empty memory parameter block.

**\$01
BCONSTAT**

BCONSTAT returns the input status of a peripheral device. This is the same as using the INP?(device) function of GFA BASIC.

X = BIOS(1,W:device)

device A word containing the code for the desired device.

<u>Number</u>	<u>Device</u>
0	Printer
1	Auxiliary Port (RS-232)
2	Keyboard
3	Midi Port
4	IKBD (output only)
5	Screen (output only)

X This function returns -1 if at least one character is ready to be handled. Zero if no characters are ready.

**\$02
BCONIN**

Retrieves a character from a peripheral device. This function is the same as INP(device) in GFA BASIC.

X = BIOS(2,W:device)

device A word length number representing the code number for the input device from which the byte of data is to be retrieved.

<u>Number</u>	<u>Device</u>
1	Auxiliary Port (RS-232)
2	Keyboard
3	Midi Port

X A word containing the data returned by the function.

**\$03
BCONOUT**

Sends a character to a peripheral device. This function is the same as OUT(device) in GFA BASIC.

VOID BIOS(3,W:device,W:character)

device A word length parameter representing the code for the device to which the character is to be sent.

<u>Number</u>	<u>Device</u>
0	Printer
1	RS-232 Port
2	Keyboard
3	MIDI Port
4	IKBD (output only)
5	Screen (output only)

character Word length parameter containing the character or data to be sent to the output device.

**\$04
RWABS**

Reads or writes data to or from a disk drive.

X = BIOS(4,W:rdwr_flag,L:buf_adr,W:sec,W:fsec,W:dev)

rdwr_flag Word length parameter setting the read/write flag.

0	Read
1	Write
2	Read *
3	Write *

* 2 and 3 allow a formatter to read and write, allowing BIOS to recognize formatted disk media change.

buf_adr Long word parameter containing the buffer address. This function works best if the address of the buffer is an even value.

sec Word length parameter for number of sectors to transfer.

fsec Word length parameter of logical number of first sector to transfer.

dev Word length parameter representing the code for the device to be written to.

<u>Code</u>	<u>Device</u>
0	Floppy Drive A
1	Floppy Drive B
2+	Disks, networks, etc.

X A negative value is returned if an error is encountered. All is well when X is equal to zero.

\$05 SETEXC

Get or set an exception vector.

X = BIOS(5,W:*vec_num*,L:*vec_addr*)

vec_num A word length parameter referring to the Vector number to set/get. (refer to Table E-1)

vec_addr A long word referencing the address to which the exception vector should be set. A value of -1 indicates that the vector address is to be read, rather than set.

X Returns either the previous address if it is setting the vector, or the current address if the function is reading the address of the vector.

Table B-1
Exception Hardware Vectors

<u>Decimal</u>	<u>Hexidecimal</u>	<u>Vector</u>
0	\$000	Reset initial SSP value
4	\$004	Reset initial PC address
12	\$00C	Address Error
16	\$010	Illegal Instruction
20	\$014	Divide by zero error
24	\$018	CHK instruction
28	\$01C	TRAPV instruction
32	\$020	Privilege violation
36	\$024	Trace Mode
40	\$028	Line A routine pointer
44	\$02C	Line 1111 (used by AES)
48	\$030	Unassigned
52	\$034	Coprocessor violation (MC68020)
56	\$038	Format error (MC68020)
60	\$03C	Uninitialized interrupt vector
64	\$040	Unassigned
*	*	
*	*	
*	*	
*	*	
*	*	
*	*	
82	\$05C	Unassigned
96	\$060	Spurious interrupt
100	\$064	Int level 1
104	\$068	Int level 2 Horizontal Blank Sync
108	\$06C	Int level 3 Normal processor interrupt level
112	\$070	Int level 4
116	\$074	Int level 5
120	\$078	Int level 6 MK68901 MFP interrupts
124	\$07C	Int level 7 Non maskable interrupt
128	\$080	Trap #0
132	\$084	Trap #1 GEM DOS interface calls
136	\$088	Trap #2 Extended GEM DOS calls
140	\$08C	Trap #3
144	\$090	Trap #4
148	\$094	Trap #5
152	\$098	Trap #6
156	\$09C	Trap #7
160	\$0A0	Trap #8
164	\$0A4	Trap #9
168	\$0A8	Trap #10
172	\$0AC	Trap #11
176	\$0B0	Trap #12
180	\$0B4	Trap #13 BIOS Calls
184	\$0B8	Trap #14 XBIOS calls
188	\$0BC	Trap #15

192	\$0C0	
*	*	
*	*	
*	*	Unassigned (reserved)
*	*	
*	*	
252	\$0FC	
256	\$100	Parallel port interrupt 0
260	\$104	RS-232 carrier detect interrupt 1
264	\$108	RS-232 clear to send interrupt 2
268	\$10C	Graphics bit done interrupt 3
272	\$110	RS-232 baud rate generator (Timer D)
276	\$114	200 Hz system clock (Timer C)
280	\$118	Keyboard/MIDI (6850) interrupt 4
284	\$11C	Polled FDC/HDC interrupt 5
288	\$120	Horizontal blank counter (Timer B)
292	\$124	RS-232 transmit error interrupt
296	\$128	RS-232 transmit buffer empty interrupt
300	\$12C	RS-232 receive error interrupt
304	\$130	RS-232 receive buffer full interrupt
308	\$134	User/Application (Timer A)
312	\$138	RS-232 ring indicator interrupt 6
316	\$13C	Polled monochrome monitor detect interrupt 7

**\$06
TICKCAL**

Returns system timer calibration, rounded to the nearest millisecond. This is the same as the TIMER function of GFA BASIC.

$X = \text{BIOS}(6)$

X Time since system was turned on (in milliseconds.)

**\$07
GETBPB**

Get pointer to BIOS parameter block for a disk drive.

$X = \text{BIOS}(7,W:\text{device})$

device Word length parameter representing the device.

<u>Number</u>	<u>Device</u>
0	Disk Drive A
1	Disk Drive B

X Pointer to BIOS parameter block.

**\$08
BCOSTAT**

Read the output status of a peripheral device. This is the same as the GFA BASIC function OUT?(device).

$X = \text{BIOS}(8,W:\text{device})$

device A word containing the code for the desired device.

<u>Number</u>	<u>Device</u>
0	Printer
1	Auxiliary Port (RS-232)
2	Keyboard
3	Midi Port
4	IKBD (output only)
5	Screen (output only)

X This function returns -1 (TRUE) if at least one character is ready to be handled. FALSE (0) if no characters are ready.

**\$09
MEDIACH**

Checks whether a disk has been changed. Actually only can determine if the media might have been changed, as this function seems to always return a value of 1.

X = BIOS(9,W:*device*)

- device* A word length parameter representing the number of the drive to be checked. This is a number between 0 and 15, with zero being drive A.
 X Value returned which represents media status.

<u>Value</u>	<u>Status</u>
0	Not changed
1	Media may have changed
2	Media has been changed

**\$0A
DRVMAP**

Returns a bitwise map of the available disk drives.

X = BIOS(&HA)

<u>Bit Map</u>	<u>Drive</u>
00000000 00000001	Disk A
00000000 00000010	Disk B
00000000 00000100	Disk C
00000000 00001000	Disk D
00000000 00010000	Disk E
00000000 00100000	Disk F
00000000 01000000	Disk G
00000000 10000000	Disk H
00000001 00000000	Disk I
00000010 00000000	Disk J
00000100 00000000	Disk K
00001000 00000000	Disk L
00010000 00000000	Disk M
00100000 00000000	Disk O
01000000 00000000	Disk P
10000000 00000000	Disk Q

**\$0B
KBSHIFT**

Checks or changes the status of the special function keys.

X = BIOS(&HB,W:*mode*)

mode A word length parameter representing the bitwise condition of the mode setting.

<u>Bits</u>	<u>Meaning</u>
0	Right shift key
1	Left shift key
2	Control key
3	Alternate key
4	Caps lock on
5	Right mouse button (Clr/Home)
6	Left mouse button (insert)
7	Reserved
-1	Retrieve status

Appendix C

XBIOS Functions



Appendix C

XBIOS Functions

This section contains a brief explanation of each of the XBIOS Functions of GEM, as it applies to GFA BASIC. A sample routine for calling each function is included. Not all of the routines are functional as listed. Some functions require preliminary set-up by other routines to work correctly.

It is not the intention of this section to provide a complete guide to using XBIOS functions, simply to provide the necessary parameters for executing XBIOS functions from GFA BASIC.

Many XBIOS functions are identical to GFA BASIC functions, and will be unnecessary for most programmers.

Each function is listed by its hexadecimal function number, followed by the accepted name for that routine. For ease of discussion, each parameter is marked as W: (word length) or L: (Long word length). The "W:" may be omitted from the function call.

NOTE:

XBIOS functions permit access to powerful system level functions of the operating system. It is assumed that you have an understanding of these functions prior to using them in your programs, or are willing to assume the risk for any dire consequences encountered.

At the very least, SAVE your program and insert a junk disk into your disk drive before attempting to run any program which may write data to your disk using an XBIOS function.

XBIOS Functions

\$00

INITMOUS

Initializes the routines for mouse packet handling.

VOID XBIOS(0,W:*mstate*,L:*addr*,L:*vector*)

mstate Word length parameter indicating mode in which mouse is to be set.

- 0 Mouse off
- 1 Mouse on (relative mode)
- 2 Mouse on (absolute mode)
- 3 Not used
- 4 Mouse on (keycode mode)

addr Long word containing the address of the 14 byte parameter block. Bytes 0-3 are used in all modes; while bytes 4-11 are used only if the mouse is in absolute mode. The following table indicates the meaning of each byte in the Mouse parameter block.

Table C-1**Mouse Parameter Block**

<u>Bytes</u>	<u>Description</u>
0	Non-zero, Y-axis zero at bottom Zero, Y-axis zero at top
1	Set mouse buttons
2	Set X-axis threshold scale delta
3	Set Y-axis threshold scale delta
4	MSB for mouse absolute maximum position on X-axis
5	LSB for mouse absolute maximum position on X-axis
6	MSB for mouse absolute maximum position on Y-axis
7	LSB for mouse absolute maximum position on Y-axis
8	MSB for mouse initial position on X-axis
9	SB for mouse initial position on X-Axis
A	MSB for mouse initial position on Y-axis
B	LSB for mouse initial position on Y-Axis

vector Long word containing address of the mouse interrupt handler.

\$01
SSBRK

Reserve to memory space, however, it must be called before initializing the operating system.

X = XBIOS(1,W:bytes)

bytes Word containing the number of bytes from the top of memory to be reserved.

**\$02
PHYSBASE**

This function returns a long word containing the starting address of the physical screen memory. It may be used for screen switching in conjunction with graphics and animation. (See Chapter 4.)

physbase% = XBIOS(2)

physbase% A long word containing the address of the start of physical screen memory.

**\$03
LOGBASE**

This function returns a long word containing the starting address of the logical screen memory. It's used primarily for screen switching in conjunction with graphics and animation. If the physical and logical screen addresses are different, the logical screen may be updated while the physical screen is displayed.

logbase% = XBIOS(3)

logbase% A long word containing the address of the start of logical screen memory.

**\$04
GETREZ**

This function returns a value corresponding with the current screen resolution.

rez% = XBIOS(4)

rez% An integer value corresponding to the state of the present screen resolution.

<u>Value</u>	<u>Mode</u>
0	Low resolution, 320 x 200 pixels, 16 colors
1	Medium resolution, 640 x 200 pixels, 4 colors
2	High resolution, 640 x 400 pixels, 2 colors

**\$05
SETSCREEN**

This function sets the screen resolution, physical address, and logical address.

VOID XBIOS(5,L:*logbase*,L:*physbase*,W:*rez*)

logbase Long word for address assigned to logical screen memory.

physbase Long word for the address assigned to the physical screen memory.

rez Word for the screen resolution. (See \$05, GETREZ)

**\$06
SETPALLETE**

Sets the screen color palette.

VOID XBIOS(6,L:*pal_ptr*)

pal_ptr A long word pointer, which points to an array of 16 words containing definitions for the color palette. The address must be an even address.

\$07**SETCOLOR**

Sets one color (number 0-15) to a specified value. Same as GFA BASIC function, SETCOLOR.

VOID XBIOS(7,W:*pal_num*,W:*colr*)

pal_num A word containing the number (0-15) of the palette register to change.

colr A word containing the code for the color in RGB format (0-7 for each digit). Color codes will range from \$000 to \$777. A negative value for this parameter causes the current color value to be returned.

\$08**FLOPRD**

Reads one or more sectors from a floppy disk.

floprd% = XBIOS(8,L:*a*,L:*b*,W:*c*,W:*d*,W:*e*,W:*f*,W:*g*)

- a* Long word pointer to a buffer large enough to hold the number of bytes in the sectors read.
 - b* Long word that must contain a zero.
 - c* Word containing number (0 or 1) of the floppy drive to read.
 - d* Word containing the sector at which to begin reading.
 - e* Word containing the track number to seek to. (0-79)
 - f* Word containing the side of the floppy to read. (0 or 1)
 - g* Word containing the number of sectors to read, not to exceed the number of sectors on the track.
- floprd%* This function returns a zero if the read was successful, or an error code if it did not succeed.

Error Codes Returned

0	No Error
-1	General error
-2	Drive not ready
-3	Unknown command
-4	CRC error
-5	Bad request, command invalid
-6	Seek error, track not found
-7	Unknown media
-8	Sector not found
-9	No paper
-10	Write error
-12	General error
-13	Write protect on
-14	Disk changed
-15	Unknown device
-16	Bad sector (during verify)
-17	Insert disk

**\$09
FLOPWR**

Write sectors to a floppy disk.

flopwr% = XBIOS(9,L:a,L:b,W:c,W:d,W:e,W:f,W:g)

- a** Long word pointer to a buffer large enough to hold the bytes to be written.
- b** Long word that must contain a zero.
- c** Word containing number (0 or 1) of the floppy drive to write to.
- d** Word containing the sector at which to begin writing.
- e** Word containing the track number to seek to. (0-79)
- f** Word containing the side of the floppy to write. (0 or 1)
- g** Word containing the number of sectors to write, not to exceed the number of sectors on the track.
- flopwr%** This function returns a zero if the write operation was successful, or an error code if it did not succeed.

Error Codes Returned

0	No Error
-1	General error
-2	Drive not ready
-3	Unknown command
-4	CRC error
-5	Bad request, command invalid
-6	Seek error, track not found
-7	Unknown media
-8	Sector not found
-9	No paper
-10	Write error
-12	General error
-13	Write protect on
-14	Disk changed
-15	Unknown device
-16	Bad sector (during verify)
-17	Insert disk

**\$0A
FLOPFMT**

Format tracks on a floppy disk.

flopfmt% = XBIOS(&HA, L:buffer, L:filler, W:device, W:sectors, W:track, W:side, W:interleave, L:magic, W:fcode)

<i>buffer</i>	Long word pointer to a buffer large enough to hold the image of an entire track.
<i>filler</i>	Long word which is presently unused, may be set to any value, but must be passed to FLOPFMT function.
<i>device</i>	Word containing the number of the device to format (0 or 1).
<i>sectors</i>	Word containing the number of sectors to format. Usually, 9 sectors per track.
<i>track</i>	Word containing the number of the track to format, between 0 and 79.
<i>side</i>	Word containing the side number to format (0 or 1).
<i>interleave</i>	Word containing the interleave factor, usually set to one.
<i>magic</i>	Long word containing the magic number, \$87654321. This number must be present.
<i>fcode</i>	Word containing code to write to each new sector formatted. Usually set to \$E5E5.

**\$0B
GETDSB**

Get device status block pointer. This is an obsolete function, there is no known use for it at this time.

VOID XBIOS(&HB)

**\$0C
MIDIWS**

Write a string to the MIDI port.

VOID XBIOS(&HC,W:*count*,L:*pointer*)

count Word containing count of number of characters to be sent.

pointer Long word for pointer to buffer containing characters to send.

**\$0D
MFPINT**

Initialize the MFP (Multi-Function Peripheral) interrupt. Allows programmer to trap a hardware interrupt.

VOID XBIOS(&HD,W:*intpt*,L:*vector*)

intpt Word containing the number of the interrupt to be set.

vector Long word address of new interrupt routine.

Table C-2
Hardware Interrupts

<u>Number</u>	<u>Definition</u>
0	I/O Port bit 0
1	undefined
2	undefined
3	undefined
4	Timer D, RS-232 baud rate generator
5	Timer C, System 200 Hz clock
6	I/O Port bit 4
7	undefined
8	Timer B
9	RS-232 Transmit error
10	RS-232 Transmit buffer empty
11	RS-232 Receive error
12	RS-232 Receive buffer full
13	Timer A, user programmable
14	I/O Port bit 6
15	I/O Port bit 7

\$OE
IOREC

Returns a pointer to a serial device's input buffer record.

iorec% = XBIOS(&HE,W:device)

device A word representing the code for the serial device.

<u>Code</u>	<u>Device</u>
0	RS-232 port
1	Keyboard
2	MIDI port

iorec% Pointer to I/O Record buffer

The record buffer consists as follows:

Long Word (4 bytes)	Pointer to Input buffer
Word (2 bytes)	Size of input buffer
Word (2 bytes)	Head Index
Word (2 bytes)	Tail Index
Word (2 bytes)	Low water mark
Word (2 bytes)	High water mark

The input buffer is a circular buffer. The Head index specifies the next write position, and the tail index specifies from where the buffer can be read. If the head and tail indices are the same, the buffer is empty.

The low and high water marks are used in connection with the RS-232 status for communications, sending the appropriate XON or CTS signal.

\$0F RSCONF

Configure the RS-232 (serial) port.

```
VOID XBIOS(&H15,W:speed,W:flow,W:ucr,W:rsr,W:tsr,W:scr)
```

speed A word length parameter that sets the baud rate as illustrated in the Table C-3.

Table C-3
Baud Rate

<u>Byte</u>	<u>Baud Rate</u>
0	19200
1	9600
2	4800
3	3600
4	2400
5	2000
6	1800
7	1200
8	600
9	300
10	200
11	150
12	134
13	110
14	75 (from Atari documentation, actually sets to 120)
15	50 (from Atari documentation, actually sets to 80)

flow

A word length parameter which sets the flow control.

<u>Value</u>	<u>Meaning</u>
0	None (the default setting)
1	XON(^S)/XOFF(^Q)
2	Request to send/clear to send(RTS/CTS)
3	XON/XOFF and RTS/CTS

ucr

Word length bit-wise parameter which sets the USART (Universal Synchronous-Asynchronous Receiver/Transmitter) control register, as illustrated by Table C-4.

Table C-4
UCR Bit-Map Settings

<u>Bit</u>	<u>Meaning</u>
0	Not used
1	0 odd parity 1 even parity
2	0 no parity 1 parity as set in bit 1
3,4	Start/Stop bits and format 00 synchronous, start 0, stop 0 10 asynchronous, start 1, stop 0 01 asynchronous, start 1, stop 1.5 11 asynchronous, start 1, stop 2
5,6	Word Length: 00 8 bits 10 7 bits 01 6 bits 11 5 bits
7	0 Use frequency from transmit control and receive control directly. 1 Divide frequency by 16

rsr

A word length bit-wise parameter that controls the receive status register.

Table C-5
RSR Bit Map Settings

<u>Bit</u>	<u>Meaning</u>
0	Enable reception
1	In synchronous mode, enable comparison of character in SCR with character in receive buffer.
2	In synchronous mode, signal that character identical to character in SCR may be received. In asynchronous mode, signal reception of start bit.
3	In synchronous mode, signal that character identical to character in SCR has been received. In asynchronous mode, signal reception of BREAK.
4	Signal frame error: Stop bit is NULL, but byte received is not.
5	Signal parity error
6	Signal buffer overrun
7	Signal buffer full

tsr A word length bit-wise parameter that controls the transmitter status register as illustrated in Table C- 6.

Table C-6
TSR Bit Map Settings

<u>Bit</u>	<u>Meaning</u>								
1	Enable transmission								
2,3	High or low output mode: <table border="0" style="margin-left: 20px;"> <tr> <td>00</td><td>High</td></tr> <tr> <td>10</td><td>High</td></tr> <tr> <td>01</td><td>Low</td></tr> <tr> <td>11</td><td>Loop back mode</td></tr> </table>	00	High	10	High	01	Low	11	Loop back mode
00	High								
10	High								
01	Low								
11	Loop back mode								
3	In synchronous mode, not used. In asynchronous mode, sends BREAK								
4	Send EOT (End Of Transmission) character after current character.								
5	Switch to reception immediately after end of transmission.								
6	Send character in sender floating register before writing new character into send buffer.								
7	Buffer empty.								

scr A word length parameter that initializes the synchronous character register. This variable should always be set to zero.

Note: Setting ucr, rsr, tsr, or scr to -1 will cause it to be ignored by TOS, and any parameters in these registers will remain unchanged.

\$10 **KEYTBL**

Sets the keyboard translation tables. Each key generates three scan codes, one in normal mode, one in shifted mode, and one in caps-lock mode. The scan code is then translated into an ASCII character from the appropriate table.

VOID XBIOS(&H10,L:*unshift*,L:*shift*,L:*caplock*)

unshift A long word pointer to the 128 byte translation table for the normal, unshifted key.

shift A long word pointer to the 128 byte translation table for a shifted key.

caplock A long word pointer to the 128 byte translation table for the key pressed with caps lock pressed.

\$11 **RANDOM**

Returns a 24-bit pseudo-random number.

***r%* = XBIOS(&H11)**

r% A 24-bit pseudo-random number.

**\$12
PROTOBT**

Creates a prototype boot sector.

VOID XBIOS(&H12,L:buf,L:ran,W:type,W:flag)

buf A long word pointer containing the address of a 512 byte buffer. This buffer may contain the image of a boot sector.

ran A long word parameter containing the serial number that will be stamped into the boot sector. Setting this value to -1 leaves the boot sectors serial number unchanged. Setting the value to any number greater than \$1000000 creates a random serial number.

type A word length parameter that contains the type of disk being worked with:

- 0 40 tracks, single sided
- 1 40 tracks, double sided
- 2 80 tracks, single sided
- 3 80 tracks, double sided

flag A word length parameter containing a flag which indicates whether the boot sector is executable or non-executable.

- 0 Non-executable
- 1 Executable
- 1 Retains current type

**\$13
FLOPVER**

Verify that a sector on a floppy disk can be read.

flopver% = BIOS(&H13, L:buffer, L:filler, W:device, W:sector, W:track, W:side, W:count)

buffer A long word pointer to a 1024 byte buffer where a list of bad sectors (if any are encountered) will be stored.

filler A long word, presently not used. Usually contains a zero, but could contain anything.

device A word length parameter for the disk to be verified.

0 for disk A
1 for disk B

sector A word length parameter representing the number of the sector to read (1-9).

track A word length parameter representing the track on which to seek the sector (0-79).

side A word length parameter for the side of the disk to read (Zero or one).

count A word length parameter for the number of sectors to read. This number can be no greater than the number of sectors on a track.

flopver% An error code is returned from this function.

Error Codes Returned

<u>Error Code</u>	<u>Meaning</u>
0	No Error
-1	General error
-2	Drive not ready
-3	Unknown command
-4	CRC error
-5	Bad request, command invalid
-6	Seek error, track not found
-7	Unknown media
-8	Sector not found
-9	No paper
-10	Write error
-12	General error
-13	Write protect on
-14	Disk changed
-15	Unknown device
-16	Bad sector (during verify)
-17	Insert disk

\$14
SCRDMP

Dump monochrome screen to printer. Not very useful. GFA BASIC function, HARDCOPY, does a screen dump in any resolution mode.

VOID XBIOS(&H14)

**\$15
CURSCONF**

Set or get cursor configuration.

cursconf% = XBIOS(&H15,W:*function*,W:*rate*)

function A word length parameter which sets the cursor function.

<u>Value</u>	<u>Meaning</u>
0	Hide the cursor
1	Show the cursor
2	Set the cursor to blink
3	Set the cursor to not blink
4	Set the cursor blink rate
5	Get the cursor blink rate
6	Not used
7	Not used

rate A word length parameter which sets the cursor blink rate, if function is set to 4. This number should be proportional to the normal blink rate. (60 Hz for color systems, 70 Hz for monochrome.) setting rate to 30 will cause the cursor to blink twice each second on a color monitor.

cursconf% Returns the current cursor blink rate, if function was set to 5. Otherwise, the value returned means nothing.

**\$16
SETTIME**

Sets the time and date for the IKBD (intelligent keyboard device). Not very useful for GFA BASIC programmers, as the GFA BASIC function SETTIME is available.

VOID XBIOS(&H16,L:*datetime*)

datetime A long word parameter containing the 32 bit mask indicating the code for the date and time.

<u>Bits</u>	<u>Sets:</u>
0-4	Number of two second increments (0-29)
5-10	Number of minutes (0-59)
10-15	Number of hours (0-23)
16-20	Day of the month (1-31)
21-24	Month (1-12)
25-31	Year (0-119, 0 indicates 1980)

**\$17
GETTIME**

Get the IKBD time and date. Again, not very useful. The same information is available from the GFA BASIC system variable, TIME\$.

gettime% = XBIOS(&H17)

gettime% A long word parameter containing the 32 bit mask indicating the code for the date and time.

<u>Bits</u>	<u>Sets:</u>
0-4	Number of two second increments (0-29)
5-10	Number of minutes (0-59)
10-15	Number of hours (0-23)
16-20	Day of the month (1-31)
21-24	Month (1-12)
25-31	Year (0-119, 0 indicates 1980)

**\$18
BIOSKEY**

Restores power up keyboard setting. (If XBIOS \$10 was used to define a new keyboard layout.)

VOID XBIOS(&H18)

\$19**IKBDWS**

Sends a string to the IKBD (Intelligent Keyboard Device). In GFA BASIC, OUT 4,x is a better method for sending a byte to the keyboard, while using PRINT# with OPEN assigning the IKBD is an easier method for sending a string.

VOID XBIOS(&H19,W:*number*,L:*ptr*)

number A word parameter of a number which is one less than the number of characters in the string.

ptr A long word pointer to the address of the string to be sent to the IKBD.

\$1A**JDISINT**

Disable a MK68901 interrupt. Refer to Table C-2 for interrupt numbers.

VOID XBIOS(&H1A,W:*int_num*)

int_num A word length parameter containing the number of the interrupt to disable. (Refer to Table C-2 for a list of applicable hardware interrupts.)

\$1B**JENABIN**

Enable a MK68901 interrupt. Refer to Table C-2 for interrupt numbers.

VOID XBIOS(&H1B,W:*int_num*)

int_num A word length parameter containing the number of the interrupt to enable. (Refer to Table C-2 for a list of applicable hardware interrupts.)

\$1C
GIACCES

Read or write to a sound chip register. For write operations, GFA BASIC programmers may use the SOUND function.

giacces% = XBIOS(&H1C,W:data,W:register)

- data* A word length parameter containing the 8-bit value to be passed to the sound chip. If read mode is selected by the value in *register*, this value is ignored.
- register* A word length parameter for the number for the register being accessed. Bit 7 of this variable indicates whether the register is to be written to or read. Zero indicates read, while a one indicates write.

\$1D
OFFGIBT

Clears the sound chip's A register. This register controls the disk drives.

VOID XBIOS(&H1D,W:bitnumber)

- bitnumber* A word length bit-wise operation which may be used to selectively clear a bit, as indicated in Table C-7. This function performs an AND of the register with the value contained in this variable.

Table C-7
Port A Registers

<u>Bit</u>	<u>Controls</u>
0	Side of floppy disk (0 or 1)
1	Drive A selected when bit is clear
2	Drive B selected when bit is clear
3	RS-232 RTS line
4	RS-232 DTR line
5	Centronics data strobe
6	General purpose output video connector
7	Unused

**\$1E
ONGIBIT**

Sets the sound chip's A register. This register controls the disk drives.

VOID XBIOS(&H1D,W:*bitnumber*)

bitnumber A word length bit-wise operation which may be used to selectively set a bit, as indicated in Table C-7. This function performs an OR of the register with the value contained in this variable.

**\$1F
XBTIMER**

Initialize the MFP timer.

VOID XBIOS(&H1F,W:*timr*,W:*ctrl*,W:*byte*,L:*ptr*)

timr A word length value between 0 and 3.

<u>Value</u>	<u>Meaning</u>
0	Timer A, user applications
1	Timer B, Graphics
2	Timer C, System timer
3	Timer D, RS-232 baud rate

ctrl A word length parameter which sets the timer's control register.

byte A word length parameter which contains the data to be written to the timer's data register.

ptr A long word parameter which points to an interrupt handler.

\$20
DOSOUND

Set sound parameters.

VOID XBIOS(&H20,L:*ptr*)

ptr A long word pointer to the address of a string of sound commands. Each sound command consists of an 8-bit hexadecimal number, followed by one or more characters.

<u>Command</u>	<u>Description</u>
\$00-\$0F	Load registers 0-15 (\$0-\$F) with one byte of data.
\$80	Write a one character argument to a temporary register.
\$81	Three one-character arguments which take the character loaded into the temporary register with the \$80 command, and control its execution. The first argument is the number of the register into which the previously stored character is to be loaded. The second argument is a two's compliment value that is added to the character in the temporary register. The third argument is an end point.
\$82-\$FF	A one byte argument. If the argument is zero, the sound is stopped. Any value greater than zero indicates the number of timer ticks that must pass before the next sound process is performed, setting how long a tone is sustained.

**\$21
SETPRT**

Get or set the printer configuration.

setprt = XBIOS(&H21,W:*config*)

config A word length parameter that is a 16-bit map configuring the printer port. If this value is set to -1, the printer port configuration is read. The configuration is set according to the following table:

<u>Bit</u>	<u>Meaning</u>
\$01	Daisywheel printer
\$02	Monochrome printer
\$04	If set, Epson dot matrix printer If not set, Atari printer
\$08	If set, Near letter Quality mode If not set, Draft mode
\$10	If set, use serial port If not set, use printer port
\$20	If set, use single sheets If not set, use fanfold paper

setprt Returns current configuration, except when config is equal to -1, the value is then meaningless.

**\$22
KBDVBAS**

Returns a pointer to the keyboard vectors.

kbdvbas = XBIOS(&H22)

**\$23
KBRATE**

Sets or gets the keyboard repeat rate.

kbrate = XBIOS(&H23,W:*delay*,W:*rate*)

- delay* A word value indicating the number of timer ticks to delay before starting to repeat the key pressed. If set to -1, the value presently stored is not changed.
- rate* A word value that sets the number of timer ticks to delay between key repeats. If set to -1, the value is not changed.
- kbrate* An integer number is returned. The number of timer ticks to delay before starting to repeat a key is returned in the high byte, while the repeat rate is returned in the low byte.
-

**\$24
PRTBLK**

Sends screen memory to the printer. Not needed in GFA BASIC. Programmers may use HARDCOPY instead.

prtblk = XBIOS(&H24,L:*pblock*)

- pblock* A long word pointer to the address of a parameter block.
-

**\$25
VSYNC**

Wait for next vertical blank. Another unnecessary command for GFA BASIC programmers. VSYNC is implemented as a part of GFA BASIC.

VOID XBIOS(&H25)

**\$26
SUPERX**

Execute code in supervisor mode.

VOID XBIOS(&H26,L:*address*)

address A long word pointer to the address of an assembly language routine to be executed in supervisor mode. (Code must end with a RTS.)

**\$27
PNTAES**

Turn off AES (Application Environmental Services). Generally, NOT a good idea in GFA BASIC, and will usually result in memorable crashes.

VOID XBIOS(&H27)

Appendix D

GEMDOS Functions



Appendix D GEMDOS Functions

This section contains a brief explanation of each of the GEMDOS Functions, as it applies to GFA BASIC. A sample routine for calling each function is included, although many GEMDOS functions are duplicated through GFA BASIC commands and functions. Where applicable, the GFA BASIC equivalent commands are listed.

Not all of the routines are functional as listed. Some functions require preliminary set-up by other routines to work correctly.

It is not the intention of this section to provide a complete guide to using GEMDOS functions, simply to provide the necessary parameters for executing GEMDOS functions from GFA BASIC.

Many GEMDOS functions are identical to GFA BASIC functions, and will be unnecessary for most programmers.

Each function is listed by its hexidecimal function number, followed by the accepted name for that routine.

For ease of discussion, each parameter is marked as W: (word length) or L: (Long word length). The "W:" may be omitted from the function call.

NOTE: GEMDOS functions permit access to powerful system level functions of the operating system. It is assumed that you have an understanding of these functions prior to using them in your programs, or are willing to assume the risk for any dire consequences encountered.

At the very least, SAVE your program and insert a junk disk into your disk drive before attempting to run any program which may write data to your disk using a GEMDOS function.

P_TERM

\$00

End application and return to the program from which the program was called.

VOID GEMDOS(&H0)

GFA BASIC Equivalent: END

C_CONIN**\$01**

Read character from the keyboard and echo the character to the screen. This function waits for a key to be pressed. Note: pressing <Control><C> can cause problems.

c_conin = GEMDOS(&H1)

GFA BASIC Equivalent: INP(2)

C_CONOUT**\$02**

Write character to standard output device (screen).

VOID GEMDOS(&H2,W:*byte*)

byte Word length parameter containing character to be displayed.

GFA BASIC Equivalent: OUT(x,*byte*)

C_AUXIN**\$03**

Read character from the RS-232 port.

c_auxin = GEMDOS(&H3)

GFA BASIC Equivalent: INP(x)

C_AUXOUT**\$04**

Output character to RS-232 port.

```
VOID GEMDOS(&H4,W:byte)
```

byte Word length parameter containing data to be transmitted via the RS-232 port.

GFA BASIC Equivalent: OUT(x,byte)

C_PRNOUT**\$05**

Simplest method of sending a character to an attached line printer connected to the Printer Port. One character is printed with each call.

```
VOID GEMDOS(&H5,W:byte)
```

byte Word length parameter containing the ASCII code of the character to be printed.

GFA BASIC Equivalent: OUT 0,byte

C_RAWIO**\$06**

Mixture of screen output and keyboard input. Reads raw characters from the keyboard and writes the characters to the screen. Characters such as <Control><C>, <Control><S>, etc., are ignored. If a key is pressed, a value is returned, otherwise a null is returned.

```
VOID GEMDOS(&H6)
```

C_RAWCIN**\$07**

Reads a raw character from the keyboard and returns it to the calling program. Characters such as <Control><C>, <Control><S>, etc., are ignored.

byte = GEMDOS(&H7)

byte Word length parameter containing the ASCII code of the character to be printed.

C_NECCIN**\$08**

Read a character from the keyboard and return value to the program. Character is not displayed on the screen. Characters such as <Control><C>, <Control><S>, etc., are ignored.

This function is the same as \$07, C_RAWCIN.

byte = GEMDOS(&H8)

byte Word length parameter containing the ASCII code of the character to be printed.

C_CONWS**\$09**

Writes a null terminated string to the screen.

VOID GEMDOS(&H9,L:*string*)

string A long word containing the address of the string to be displayed.

GFA BASIC Equivalent: PRINT

C_CONRS**\$0A**

Read and edit a string entered from the keyboard. The cursor, backspace, and delete keys are active, permitting editing functions.

VOID GEMDOS(&HA,L:*addr*)

addr Long word containing the address of the input buffer.

GFA BASIC Equivalent: INPUT, FORM INPUT

C_CONIS**\$0B**

Check status of keyboard input.

sta = GEMDOS(&HB)

sta Returns the status of the keyboard:

0 = No character ready for input

-1 = Character ready for input

GFA BASIC Equivalent: INP?(2)

D_SETDRV**\$0E****Set current drive.****VOID GEMDOS(&HE,W:*drive*)**

drive Word length parameter containing a number between 0 and 15, representing the number of the drive to be set as the current default drive. 0 equal drive A, 1 is drive B, 2 is drive C, etc.

GFA BASIC Equivalent: CHDRIVE drive

C_CONOS**\$10****Check if console is ready to receive characters****sta = GEMDOS(&H10)**

/ **sta** Status of console.

0 = Not ready

-1 = Ready to receive

GFA BASIC Equivalent: OUT?(2) (always ready)

C_PRNOS

\$11

Check status of printer.

sta = GEMDOS(&H11)

sta Input status of printer.

 0 = Not ready

 -1 = Ready to receive a character.

GFA BASIC Equivalent: OUT?(0)

C_AUXIS

\$12

Check status of serial (RS-232) port.

sta = GEMDOS(&H12)

sta Status of port.

 0 = No characters waiting.

 -1 = Characters are waiting.

GFA BASIC Equivalent: INP?(1)

C_AUXOS**\$13**

Check if Serial (RS-232) port is ready to receive characters.

sta = GEMDOS(&H13)

sta Output status of RS-232 port.

0 = Not ready.

-1 = Ready for characters.

GFA BASIC Equivalent: OUT?(1)

C_GETDRV**\$19**

Get device number of current drive.

drive = GEMDOS(&H19)

drive Device number of current drive.

0 = Drive A

1 = Drive B

2 = Drive C

etc.

F_SETDTA**\$1A**

Set disk transfer address to address of a 44-byte buffer.

VOID GEMDOS(&H1A,L:*addr*)

addr Long word containing address of buffer.

T_GETDATE**\$2A**

Get the current date.

dte = GEMDOS(&H2A)

dte Current date returned from TOS as an integer value stored in the following manner:

<u>Bits</u>	<u>Meaning</u>
0-4	Day (1-31)
5-8	Month (1-12)
9-15	Year (0-119, 0=1980)

GFA BASIC Equivalent: *dte\$* = DATE\$

T_SETDATE**\$2B**

Set new date.

VOID GEMDOS(&H2B,W:*dte*)

dte Date to set as a word in the following format:

<u>Bits</u>	<u>Meaning</u>
0-4	Day (1-31)
5-8	Month (1-12)
9-15	Year (0-119, 0=1980)

GFA BASIC Equivalent: DATE\$ = *dte\$*

T_GETTIME**\$2C**

Get the current time from the operating system.

tme = GEMDOS(&H2C)

tme Word length parameter containing the current time as obtained from the operating system in the following format:

<u>Bits</u>	<u>Meaning</u>
0-4	Number of two second increments (0-29)
5-10	Number of minutes (0-59)
11-15	Number of hours (0-23)

GFA BASIC Equivalent: *tme\$* = TIME\$

T_SETTIME**\$2D**

Set the current time.

VOID GEMDOS(&H2D,W:tme)

tme Word length parameter containing the current time as obtained from the operating system in the following format:

<u>Bits</u>	<u>Meaning</u>
0-4	Number of two second increments (0-29)
5-10	Number of minutes (0-59)
11-15	Number of hours (0-23)

GFA BASIC Equivalent: TIME\$ = tme\$

F_GETDTA**\$2F**

Get disk transfer address.

f_getdta = GEMDOS(&H2F)

f_getdta Long word parameter containing the address set by F_SETDTA. (Used by F_SFIRST and F_SNEXT).

S_VERSION**\$30**

Returns the version number of GEMDOS in low byte-high byte format.

s_version = GEMDOS(&H30)

s_version Word length parameter containing version number.

P_TERMRES**\$31**

Terminate a process, but retain a specified number of bytes in memory. This function is not usable with GFA BASIC.

VOID GEMDOS(&H31,L:bytes,W:encode)

bytes Long word parameter specifying number of bytes to keep in memory.

encode Word length parameter containing the exit code for the process being terminated.

NOTE: Since the many of the following commands are rather difficult, and totally unneeded in GFA BASIC, GEMDOS calls have been omitted where unnecessary. The GFA BASIC equivalent commands are shown.

D_FREE**\$36**

Get amount of free space remaining on a specified disk. This command is rather difficult, and, fortunately totally unnecessary for GFA BASIC programmers.

GFA BASIC Equivalent: d_free = DFREE

D_CREATE**\$39**

Create a directory. This is another unnecessary function for GFA BASIC Programmers.

GFA BASIC Equivalent: MKDIR

D_DELETE**\$3A**

Delete a subdirectory.

GFA BASIC Equivalent: RMDIR

D_SETPATH**\$3B**

Set path for disk access.

GFA BASIC Equivalent: CHDIR

F_CREATE**\$3C**

Creates a new file. Returns file handle that is used in further operations.

GFA BASIC Equivalent: SAVE, BSAVE, OPEN

F_OPEN**\$3D**

Open a file. Returns information similar to F_CREATE.

GFA BASIC Equivalent: OPEN

F_CLOSE**\$3E**

Close a file.

GFA BASIC Equivalent: CLOSE

F_READ**\$3F**

Read a file.

GFA BASIC Equivalent: INPUT #

F_WRITE**\$40**

Write to an open file.

GFA BASIC Equivalent: WRITE #, PRINT #, OUT #

F_DELETE**\$41**

Delete a file.

GFA BASIC Equivalent: KILL

F_SEEK**\$42**

Move file pointer.

GFA BASIC Equivalent: SEEK

F_ATTRIB**\$43**

Get and set file attributes.

f_attrib = GEMDOS(&H43,L:*path*,W:*readset*,W:*attr*)

path Long word pointer to null-terminated string containing the name of the file.

readset Word length parameter containing a zero if you wish to read the file attributes, one to set the attributes.

attr Word length parameter containing the attributes:

- \$01 Read only.
- \$02 Hidden file
- \$04 Hidden system file
- \$08 File
- \$10 File is subdirectory
- \$20 File has been written and closed

f_attrib Negative value is returned if an error was encountered, otherwise it will contain the files attributes.

F_DUP**\$45**

Generates a substitute file handle. Not useful in GFA BASIC.

F_FORCE**\$46**

Force a file handle. Not useful in GFA BASIC.

D_GETPATH**\$47**

Get current directory. Another unnecessary function in GFA BASIC.

GFA BASIC Equivalent: DIR\$

M_MALLOC**\$48**

Allocate or read free memory. Again, unnecessary for GFA BASIC programmers to use.

GFA BASIC Equivalent: FREE or RESERVE

M_FREE**\$49**

Free allocated memory. Unnecessary in GFA BASIC.

GFA BASIC Equivalent: RESERVE

M_SHRINK**\$4A**

Shrink size of allocated memory. Again, not needed in GFA BASIC.

GFA BASIC Equivalent: RESERVE

P_EXEC**\$4B**

Load or execute a process. Another function duplicating a GFA BASIC Function.

GFA BASIC Equivalent: EXEC

P_TERM**\$4C**

Terminate process. Not usable in GFA BASIC.

GFA BASIC Equivalent: END, EDIT

F_SFIRST**\$4E**

Search for first occurrence of a file. Not necessary in GFA BASIC.

GFA BASIC Equivalent: EXIST

F_SNEXT**\$4F**

Search for next occurrence of a file. Not necessary in GFA BASIC.

GFA BASIC Equivalent: EXIST

F_RENAME**\$56**

Rename a file. Not necessary in GFA BASIC.

GFA BASIC Equivalent: NAME

F_DATIME**\$57**

Get or set a files date and time stamp.

VOID GEMDOS(&H57,L:*ptr*,W:*handle*,W:*sg*)

ptr Long word pointer to a string containing the date/time stamp.

handle Word length parameter containing the handle of the file.

sg Word length parameter specifying whether to get or set the date/time stamp.

0 = Set date/time stamp
1 = Get date/time stamp

Appendix E

GFA BASIC Editor/ Interpreter Quick Reference Table



Appendix E

GFA BASIC Editor/Interpreter Quick Reference Table

Refer to Chapter 1, The GFA BASIC Editor/Interpreter for a more detailed description of each editor/Interpreter command.

<u>Key Strokes</u>	<u>Function</u>
<F1>	Load
<Shift><F1>	Save
<F2>	Merge
<Shift><F2>	Save, ASCII
<F3>	Llist
<Shift><F3>	Quit
<F4>	Block
<Shift><F4>	Quit
<F5>	Block End
<Shift><F5>	Block Start
<F6>	Find
<Shift><F6>	Replace
<F7>	Page Down
<Shift><F7>	Page Up
<F8>	Insert/Overwrite toggle
<Shift><F8>	Text 8/16 toggle
<F9>	Flip to Output screen
<Shift><F9>	Switch to Output screen
<F10>	Test loops

<Shift><F10>	Run program
<Control><Left Arrow>	Moves cursor to the beginning of the line
<Left Arrow>	Moves cursor left one character
<Control><Right Arrow>	Moves cursor to the end of the line
<Right Arrow>	Moves cursor right one character
<Control><Up Arrow>	Moves cursor one page back
<Up Arrow>	Moves cursor up one line
<Control><Down Arrow>	Moves cursor ahead one page
<Down Arrow>	Moves cursor down one line
<Control><Tab>	Moves cursor left one tab position
<Tab>	Moves cursor right one tab position
<Return>	Moves cursor down one line
<Control><Home>	Moves cursor to beginning of program
<Home>	Moves cursor to beginning of the currently displayed page
<Control><Z>	Moves the cursor to the end of the program
<Backspace>	Moves the cursor left one character and deletes that character
<Delete>	The character under the cursor is deleted
<Control><Delete>	The entire line the cursor is located on is deleted

<Insert>	Moves all lines beginning with the line on which the cursor is located down one line, and allows text to be inserted into the listing
<Undo>	Reverses last changes made to a program line, restoring the line to it's original condition, providing that the cursor has not been moved off of the line
<Control><F>	Finds next occurrence of specified search string
<Control><R>	Replaces search string with specified replacement string

Index



* 30 BMOVE 51
 + 31 BOX 52
 - 31 BPUT 53
 BAS 16 BSAVE 54
 BAK 16 Baud Rate C-15
 LIST 17 Binary 10
 / 32 Blk End 18
 < 32 Blk Sta 18
 >= 33 Boolean 9
 > 34 Break and Error 29
 ABS 36 CALL 56
 ADDRIN 37 CHAIN 57
 AND 38 CHDIR 58
 ARR PTR 40 CHDRIVE 58
 ASC 41 CLEAR 62
 ATN 41 CLOSE 63
 ASCII code 41 CLOSEW 64
 Arc tangent 41 CLR 65
 Advantages 6 COLOR 66
 BCDONOUT B-4 COS 67
 BCOSTAT B-7 CONT 67
 BCOUNTIN B-3 CONTRL 67
 BCOUNTIN B-3 CURSCONF C-22
 BIOS 47 CVF 70
 BIOS\$ 46 CVS 72
 BGET 45 CVI 70
 CVL 71 CVD 69
 BIOSKEY C-23 CVL 71
 BLOAD 50 CVAUXIN D-3
 BRBLT 47 C-AUXDS D-8
 C-AUXOS D-9 C-AUXS D-8
 C-AUXIN D-3 C-AUXOUT D-4

C_CONIN D-3
C_CONIS D-6
C_CONOS D-7
C_CONOUT D-3
C_CONRS D-6
C_CONWS D-5
C_GETDRV D-9
C_NEcin D-5
C_PRNOS D-8
C_PRNOUT D-4
C_RAWCIN D-5
C_RAWIO D-4
Cartesian coordinate system 266
Cathode ray tube 264
Child directories 11
Command lines 16
Commands 26
Computer graphics 264
Constants 27
Copy 18
C
DATA 72
DATA Handling 28
DATE\$ 73
DEC 74
DEFFILL 75,272
DEFFN 78
DEFLINE 79
DEFLIST 80
DEFLIST 0 15
DEFLIST 1 15
DEFMARK 81
DEFMOUSE 82
DEFNUM 84
DEFTEXT 84
DFREE 86
DIM 86
DIM? 87
DIR 88
DIR\$ 89
DIV 90
DO...LOOP 91
DOSOUND C-27
DPEEK 91
DPOKE 92
DRAW 93
DRVMAP B-9
D_CREATE D-13
D_DELETE D-14
D_FREE D-13
D_GETPATH D-17
D_SETDRV D-7
D_SETPATH D-14
Decimal 10
Default display 15
Delete 18
Destination Memory Form Definition Block 48
Direct 21
Disk Commands 28
Dragon plot 294
EDIT 93
ELLIPSE 94
END 95
EOF 96
EQV 96
ERASE 97
ERR 98
ERROR 100
EVEN 100
EXEC 101
EXIST 101
EXIT IF 102
EXP 103
EasyTerm 477
End 18
Exception Hardware Vectors B-6
FALSE 103
FATAL 104
FIELD 104
FILES 105
FILESELECT 106
FILL 106, 272

FIX 107
FLOPFMT C-11
FLOPRD C-7
FLOPVER C-20
FLOPWR C-9
FOR...NEXT 108
FORM INPUT 109
FORM INPUT AS 110
FRAC 111
FRE 111
FULLW 112
F_ATTRIB D-16
F_CLOSE D-15
F_CREATE D-14
F_DELETE D-15
F_DUP D-16
F_GETDTA D-12
F_OPEN D-14
F_READ D-15
F_SEEK D-16
F_SETDTA D-9
F_WRITE D-15
Find 20
Flip 21
Fractal graphics 265
Fractals 293
Frank Ostrowski 4
Frequency 357
GB 112
GCTRL 113
GEMDOS 113
GEMDOS Functions D-2
GEMSYS 114
GET 114
GETBPP B-8
GETDSB C-11
GETMPB B-2
GETREZ C-5
GETTIME C-23
GENie 390
GFA BASIC Editor/Interpreter 15

GFA Editor/Interpreter 14
GFABASICRO.PRG 16
GIACCES C-25
GINTIN 116
GINTOUT 116
GOSUB 117
GOTO 118
GRAPHMODE 119
Getrez 269
Graphics Commands 28
Graphics techniques 265
HARDCOPY 120
HEX\$ 121
HIDEM 121
HIMEM 122
Hardware Interrupts C-13
Hertz 357
Hexadecimal 10
Hide 19
Hierarchical file system 11, 16
IF 122
IKBDWS C-24
IMP 123
INC 3, 124
INFOW 124
INITMOUS C-3
INKEY\$ 125
INP 126
INP? 128
INPUT 128
INPUT# 130
INPUT\$ 131
INSTR 132
INT 132
INTIN 133
INTOUT 133
IOREC C-13
Input and Output 28
Input/Output 390
Insert 21
Integer 9

-
- Ivan Sutherland 264
JDISINT C-24
JENABIN C-24
KBDVBAS C-28
KBRATE C-29
KBSHIFT B-10
KEYTBL C-18
KILL 133
Keyboard commands 22
LEFT\$ 134
LEN 134
LET 135
LINE 135, 270
LINE INPUT 136
LINE INPUT# 137
LIST 137
LLIST 138
LOAD 138
LOC 139
LOCAL 139
LOF 140
LOG 141
LOG10 142
LOGBASE C-5
LOGO 2
LPEEK 142
LPOKE 143
LPOS 143
LPRINT 144
LSET 145
Labels 7
Line numbers 7
Llist 17
Load 16
Logical operations 26
MAX 146
MEDIACH B-9
MENU 146
MENU KILL 148
MENU OFF 148
MFPINT C-12
MID\$ 152
MID\$= 153
MIDIWS C-12
MIN 154
MKD\$ 155
MKDIR 155
MKFS 156
MKIS 156
MKLS 157
MKS\$ 157
MOD 158
MONITOR 158
MOUSE 159
MOUSEK 160
MOUSEX 160
MOUSEY 161
MUL 162
M_FREE D-17
M_MALLOC D-17
M_SHRINK D-18
Mandelbrot set 296
Massachusetts Institute of Technology 264
Mathematical Functions 27
Mathematical operations 10
Memory Form Definition Block 49
Merge 17
MichTron
Modula 2 4
Mouse Commands 28
Mouse Parameter Block C-4
Move 18
NAME 163
NEOLOAD.BAS 313
NEOWRITE.BAS 311
NEW 163
NOT 164
New 18
Normalized device coordinate 267
OCT\$ 165
ODD 165
OFFGIBT C-25

ON BREAK 166
ON BREAK CONT 166
ON BREAK GOSUB 166
ON ERROR 167
ON ERROR GOSUB 167
ON MENU 168
ON MENU BUTTON 172
ON MENU IBOX 171
ON MENU KEY 169
ON MENU MESSAGE 170
ON MENU OBOX 171
ON...GOSUB 174
ONGIBIT C-26
OPEN 175
OPENW 176
OPTION 177
OPTION BASE 178
OR 179
OUT 180
OUT? 181
Octal 10
Operators 26
Ovrwrt 21
PAUSE 182
PBOX 182
PCIRCLE 183
PEEK 184
PELLIPSE 185
PHYSBASE C-5
PI 187
PLOT 187
PNTAES C-30
POINT 188
POKE 188
POLYFILL 189
POLYLINE 191
POLYMARK 192
POS 193
PRBOX 194
PRINT 195
PRINT # 197
PRINT AT 196
PRINT USING 198
PROCEDURE 200
PROTOBT C-19
PRTBLK C-29
PSAVE 201
PTSIN 201
PTSOUT 202
PUT 202
PUT# 203
P_EXEC D-18
P_TERM D-2
P_TERMRES D-13
Parent directory 11
Pascal 4
Pg down 20
Pg up 20
Pitch 358
Port A Registers C-25
Program Control Statements 27
Program lines 15
QUIT 204
Quit 17
RANDOM 204
RANDOM C-18
RBOX 206
READ 207
RELSEEK 208
REM 209
REPEAT...UNTIL 209
RESERVE 210
RESTORE 211
RESUME 212
RETURN 213
RIGHT\$ 214
RMDIR 215
RND 215
RSCONF C-14
RSET 216
RSR Bit Map Settings C-17
RUN 217

- RWABS B-4
Radians to degrees 42
Raster coordinate (RC) system 267
Real 9
Replace 19
Restore_palette 291
Root directory 11
Run 21
SAVE 217
SCRDMP C-21
SDPOKE 218
SEEK 219
SETCOLOR 219
SETCOLOR C-7
SETEXC B-5
SETPALLETE C-6
SETPRT C-28
SETSCREEN C-6
SETTIME 221, C-22
SGET 222
SGN 222
SHOWM 223
SIN 224
SLPOKE 224
SOUND 225
SPACE\$ 226
SPC 227
SPOKE 227
SPRITE 228
SPUT 230
SQR 230
SSBRK C-4
ST BASIC 2
STOP 231
STR\$ 231
STRING\$ 232
SUB 233
SUPERX C-30
SWAP 234
SYSTEM 234
S_VERSION D-12
Save 16
Save_palette 291
Sequential and Random File Handling 28
Set_color 293
Set_rez 270
Sine wave 357
Sketchpad 264
Sound 29
Sound 356
Sound Edit 363
Sound Wave 357
Source Memory Form Definition Block 48
Start 18
Statements 26
String 9
String Art 273
String Handling Commands 27
Structured programming 7
System Commands 27
System Level Commands 29
System variables 26
TAB 235
TAN 235
TEXT 237
TEXT 16 20
TEXT 8 20
TICKCAL B-8
TIME\$ 238
TIMER 238
TITLEW 239
TROFF 239
TRON 240
TRUE 241
TRUNC 241
TSR Bit Map Settings C-17
TYPE 242
T_GETDATE D-10
T_GETTIME D-11
T_SETDATE D-10
T_SETTIME D-12
Tektronics 264

Telecommunication 391

Test 21

The Fractal Geometry of Nature 296

The GFA BASIC Language 26

Timer Functions 29

UCR Bit-Map Settings C-16

UPPER\$ 243

VAL 244

VAL? 244

VARPTR 245

VDIBASE 245

VDISYS 246

VOID 246

VSYNC 247, C-29

Variable Types 9

Variable name 8

WAVE 247

WHILE...WEND 249

WINDTAB 249

WRITE 251

WRITE# 251

Windows and Menus 29

Word 26

Write 18

XBIOS 252, 269

XBIOS 269

XBIOS Functions C-2

XBTIMER C-26

XOR 253

Xmodem 484

GFA BASIC Training

ReBoot Camp

The Beginners Guide to GFA BASIC Programming



Introduction to a NEW world!

GFA BASIC TRAINING - REBOOT CAMP is your introduction to the most powerful and versatile version of BASIC available today, GFA BASIC.

GFA BASIC is the ideal language for all programmers. With its many special features and the interactive editor GFA BASIC reduces your programming time, and makes programming fun.

A Helpful Tutorial

GFA BASIC TRAINING - REBOOT CAMP is intended for all beginning programmers and provides a solid foundation in programming techniques and concepts.

This excellent tutorial for beginning programmers guides you through your first steps as a programmer, helping you past many of the frustrations of learning a new programming language and the theory that goes with it.

You, The Programmer

The chapters of **GFA BASIC TRAINING - REBOOT CAMP** will help you learn new skills as you create your own programs. From the very first chapter, *YOU* are a programmer!

GFA BASIC TRAINING - REBOOT CAMP guides you through the many features of GFA BASIC. Learn to incorporate graphics, sound, text, and mouse-control into your programs.

Features Include:

- Mouse-drawing program
- Simon game
- On-screen jokebook
- Tune-player
- Simple animated game
- Dice-tossing game
- Graphics display program
- Programming aids
- Clear, easy-to-follow instructions
- And much more!

A Special Bonus Offer

Also available as part of a special package with the purchase of **GFA BASIC TRAINING - REBOOT CAMP** is a disk with the programs from the book, graphics programming aids, and a unique GFA BASIC Reference Card.

A New Programming Aid

The GFA BASIC Reference Card outlines every command in GFA BASIC, with a description of the command, and the proper syntax for using the command. This is a unique programming aid that every GFA BASIC programmer will find indispensable.

*** Requires GFA Basic ***

\$19.95

MichTron
576 S. Telegraph, Pontiac, MI 48053
Orders and Information (313) 334-5700

GFA-BASIC Reference Card

The Indispensable Programmers Tool

Two years ago GFA-BASIC was introduced for the Atari ST. Since then it has become the best selling program for the ST with over 50,000 copies sold world wide. More than one enthusiastic ST user has stated "GFA BASIC is the language which *should* have been shipped with the ST." GFA BASIC quickly became the standard by which all others were judged.

Now, MICHTRON, Inc. has developed a new programming aid for all GFA BASIC programmers, **The GFA BASIC Reference Card**, sure to become one of the handiest tools for the thousands of users working with GFA BASIC.

The GFA BASIC Reference Card was prepared by George Miller, former Technical Editor at COMPUTE! Publications, and one of the leading experts on GFA BASIC, with consultation from the experts at GFA Systemtechnik, the authors of GFA BASIC.

Designed of durable stock, this specially treated, attractive card folds out to keep all the

information you need at your fingertips at all times.

Every command has been grouped according to function. For instance, all the graphics commands are arranged under the Graphics heading.

Each entry contains a brief description of the command, and demonstrates the proper syntax for use. No longer will it be necessary to interrupt your programming process to page through a manual, searching for the command you need.

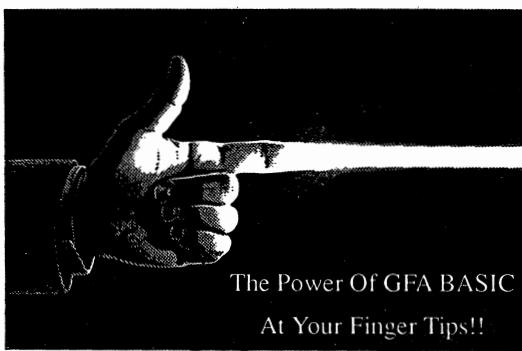
The GFA BASIC Reference Card, a programming tool that no programmer should be without. Available now from MICHTRON.

Call MICHTRON to place your order now!

Price: \$4.95

MasterCard, Visa, and C.O.D. accepted

Shipping and C.O.D. charges (where applicable) extra



The GFA BASIC Programmers Reference Guide, Vol. I

Here's the information that every GFA BASIC programmer has been waiting for! Packed with the answers to the most commonly asked questions, this book is sure to become one of your most valuable sources for programming information.

Unlock the power of GFA BASIC with the detailed explanations of every command, and the invaluable information contained in the Appendices: BIOS, GEMDOS, and XBIOS explained in detail.

Topics explained include:

- Graphics and Animation
- Sound and Music
- Menus
- Programming the Peripheral Ports
- *And much more!*

Demonstration programs are included which illustrate the use of each command, as well as several useful utility programs:

<i>Shape Editor</i>	Create objects for animation sequences.
<i>Sound Edit</i>	Experiment with sounds to achieve just the effect you need.
<i>EasyTerm</i>	A full-featured terminal program.

Plus many more sample programs!

❖ Requires the **GFA BASIC Interpreter** ❖

