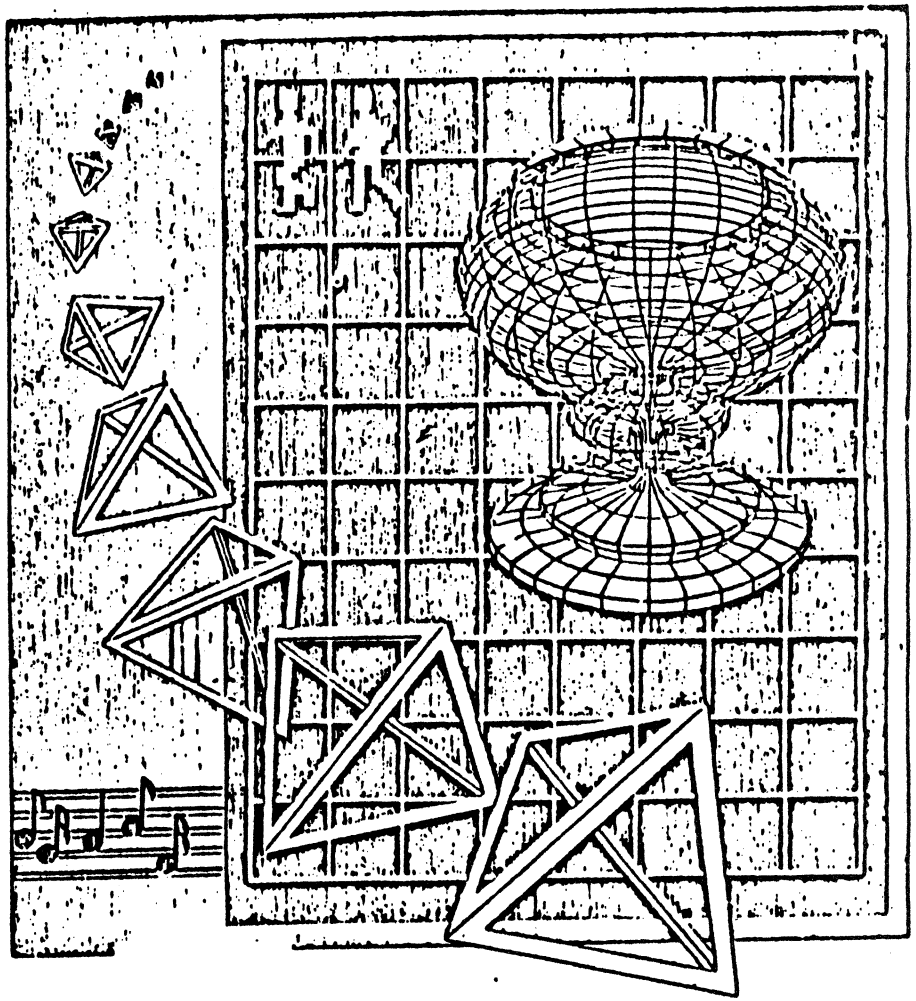
MMSoft®
10175 S.W. Barbur Blvd.
Suite 202B
Portland, OR 97219
(503) 244-4181

MMSoft®

GraFORTH

Graphics Language Reference Manual



GraFORTH

by Paul Lutus

GraFORTH LANGUAGE MANUAL

Notice

Insoft and Paul Lutus reserve the right to make improvements in the product described in this manual at any time and without notice.

Disclaimer of all Warranties And Liabilities

Insoft Company and Paul Lutus make no warranties, either expressed or implied, with respect to the software described in this manual, its quality, performance, merchantability or fitness for any particular purpose. This software is licensed "as is". The entire risk as to the quality and performance of the software is with the buyer. Should the software prove defective following its purchase, the buyer (and not INSOFT Company, or Paul Lutus, their retailers or distributors) assumes the entire cost of all necessary servicing, repair or correction and any incidental or consequential damages. In no event will INSOFT Company, or Paul Lutus be liable for direct, indirect, incidental or consequential damages resulting from any defect in the software even if they have been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

The word Apple and the Apple logo are registered trademarks of Apple Computer.

Apple Computer, Inc. makes no warranties, either expressed or implied, regarding the enclosed computer software package, its merchantability or its fitness for any particular purpose.

TABLE OF CONTENTS

CHAPTER ONE: PREVIEW

CHAPTER TABLE OF CONTENTS:

Introduction to GraFORTH

The Apple computer has some potentially powerful graphics capabilities. One of the most impressive of these is the presence of high-resolution color graphics. While there has been a large number of programs written which use this capability, sometimes in a most dramatic way, and there have been several outstanding graphics utilities written to ease the task of adding Apple Graphics to programs, until now, no computer languages have been specifically created for the purpose of fully exploiting these features. GraFORTH is just such a language.

A Family of Languages

GraFORTH is the latest member of a powerful new "family of languages" developed for Insoft by Paul Lutus. The first of these related languages to be released was TransFORTH. While TransFORTH and GraFORTH are related, each of these languages has different functions and capabilities, and is designed to meet different needs. They are related in the ways members of a family are related - they have the same parentage, that of the FORTH language. In a moment, we'll take a look at that heritage, and discuss the differences between GraFORTH and other FORTH implementations. But first, let's look at the capabilities of GraFORTH you'll very soon be learning!

Features

GraFORTH provides many features not seen before on small computers. The system can draw three-dimensional images, in color, at rates that make animation possible. A sophisticated music synthesizer, a part of the language, allows the addition of music as well as sound to GraFORTH programs. Text display may be in any size, color, or typeface, and mixed with graphics images on any part of the screen. Personalized character fonts may be created, and fonts full of different two-dimensional images may be block printed to any screen location under full program control. Clearly, this is a programming language designed for applications where fast, sophisticated graphics capability is important, such as the development of games and entertainment software.

Comparison with Standard FORTH

The above features are embodied in a very fast, fully compiled version of FORTH. Nearly all other Apple languages (both BASICs, UCSD Apple Pascal, Apple FORTRAN, and most other FORTHs) are interpreted while they are running. This is often done to provide what is called 'code transportability', the ability to take programs from one computer and run them on another with either no or few modifications. Unfortunately, this drastically reduces the speed of your programs. GraFORTH (and TransFORTH) have been designed for the computer you own, the Apple. They have been specifically written to make maximum use of the features built into your machine, and therefore no attempt has been made to create transportable code. By compiling directly to 6502 machine language, speed was greatly increased over nearly every other language - a must for smooth, fast, animation quality graphics. Even though GraFORTH is fully compiled for the purpose of increased speed, commands may still be typed directly at the keyboard, rather like an interpreted language. As implemented, then, GraFORTH has both the speed of a compiled language and the immediate feedback of an interpreted language, the best of both worlds. Finally, GraFORTH, unlike standard implementations of FORTH, uses standard Apple DOS commands and file structures, to retain compatibility with the work you have already done with your computer, and to reduce the time it will take to learn GraFORTH.

If you are already familiar with another version of FORTH, you will find many similarities and many differences between GraFORTH and other FORTH versions, as GraFORTH is only loosely related to these other languages. The general structure of the language has been retained (at least outwardly), but the implementation of that structure is vastly different. These changes have been made for very specific reasons. In short, the intended usage of GraFORTH is very different from that for which FORTH was originally designed. GraFORTH is a computer graphics language, and this in and of itself brought about many changes. Further, it was our intention to make GraFORTH as easy to learn and as similar to existing Apple environments as possible. Therefore, if you already know FORTH, we hope you will bear in mind that this language has been designed for those who do not share your knowledge of FORTH-like environments and who want a fast, easy to learn graphics language. For those of you who do not know FORTH, dive in! You will find GraFORTH to be a powerful, yet intuitive language. Very soon you will be using your Apple to do things you never thought were possible before!

Comparison with TransFORTH

By way of contrast, while GraFORTH is a powerful graphics programming language, restricted to whole number (integer) calculations for the purpose of graphics speed, TransFORTH is a scientific and business oriented language with floating-point arithmetic and a much more extensive operating system. TransFORTH also has two-dimensional line-drawing and TURTLEGRAPHICS capabilities, but no three-dimensional graphics, and character graphics are limited to selection of pre-defined character sets. Thus, TransFORTH has much more calculating ability, but less graphics, while just the opposite is true of GraFORTH.

Program Editing and Storage

Programs, subroutines, or 'words', as they are known in FORTH, can be written in the language editor and stored in text files for later modification or use. Because these files are standard DOS text files, any editor of the user's choosing which creates such files may be used. Because program segments may be saved in this way, the accumulation of proven program modules is encouraged, which in turn encourages the practice of good programming techniques.

Manual Overview

Structure

The text portion of this manual is divided into three parts - an introductory or context-setting section (Chapters 1 and 2), a tutorial-based content section of seven chapters to help you understand and put to use the GraFORTH language system (3 through 9), and a section of appended reference material, including the GraFORTH Word Library Listings, Technical Data, and Index. Throughout these chapters, diagrams are used to support the text. These illustrations and the abundant use of headings should make it possible for you to skim the text, get a sense of the subject matter, find general topic areas in the body of text, and never lose your sense of where you are. The Index should help you find specific topics quickly.

Review of Content

Part II, the content of the manual (that is, that material which is about the language itself) is presented in seven major chapter divisions. Chapter 3 is primarily an introduction to the FORTH language aspects of GraFORTH, including an explanation of the definition of words, stack operation, and control structures. (In addition to being a good introduction to GraFORTH, much of the material covered in this chapter pertains to other FORTHs as well, making it an excellent FORTH overview.) Chapter 4 covers text entry, special characters, and the supplied text editor. It shows how to write and modify GraFORTH programs or "words" and how to compile them into memory from the editor buffer or from disk. Chapter 5 presents extended GraFORTH capabilities and describes how it operates, how it relates to and uses the DOS 3.3 disk operating system, and how its data structures - variables and strings - are created and used. Chapter 6 introduces GraFORTH's two-dimensional graphics capabilities including plotting and line drawing, color selection/filling, and the TURTLEGRAPHICS commands. Chapter 7 describes character graphics, particularly a program called CHAREDITOR, which allows the design of new character fonts and images that can be block printed to the screen. Chapter 8 reveals the GraFORTH 3-D graphics system, including moving and manipulating objects in 3-D space. The program IMAGEDITOR, which allows the creation and modification of 3-D objects, and another, called PROFILE, which speeds up the process for the particular class of objects which rotate or revolve around a central axis, are introduced. Another program, named PLAY, winds up the discussion of 3-D graphics by allowing you to "play" with an object in space, as you will discover in a short exercise at the end of this chapter. Chapter 9 describes how to add music (as opposed to sounds) to your programs, and Chapter 10 concludes Part II with a discussion of marketing software developed using GraFORTH. That's a lot of content, which you surely must be eager to get to, but first perhaps we should talk about the manual for a bit.

How to Use This Manual

Differences of Style

It is important to realize that everyone uses manuals according to his or her own individual learning styles and skill levels. There are those of us who start from the beginning and carefully read every word, and there are others who bound ahead looking for just enough information to "get on with it". Still others like to live on the edge, boot the disk first, and only use the manual if they have to look something up later. Furthermore, even the same reader will have differing moods and levels of interest, and will use a technical manual in different ways at different times according to his or her current understanding of the product.

Tutorial Learning

This manual is set up to be, first of all, a tutorial to guide you gradually through the steps you need to take to learn the GraFORTH language and begin to put it to use. 'Tutorial learning' has become the primary method of microcomputer instruction. Actually, it's a bit of a misnomer. There is really no tutor, unless a technical manual can be considered such. For the most part, it will be just you and the manual and whatever other resources you can pull together. Be advised, however, that there are many differences between GraFORTH and other FORTH implementations. Because of these differences (we think of them as improvements), we advise you, even if you know FORTH already, to read the manual carefully at the beginning.

Later, of course, you will be using the manual more as a reference guide than as a tutorial, and will need to be able to find specific items of information quickly. There is nothing more frustrating than knowing that you saw something someplace, but can't quite remember where. We'll help you find it, after all, you may be living with this manual for a few weeks. In either case, tutorial or reference, we have tried to accommodate all styles of learning.

Reference Aids:

Multiple Tables of Contents

As mentioned above, there are various reference aids which should allow you to find what you want quickly when using the manual as a reference guide. At the beginning of the manual, there is a comprehensive table of contents which presents the major topics of the manual, with page numbers, in the order in which they appear. Each chapter has a similar, but more complete table of contents for that chapter.

The Word Library Definitions List

Appendix A, in the back of the manual, contains an alphabetically arranged list of annotated definitions of all the GraFORTH words which come with the system. Because this is an important source of information about the language to which you will be referring frequently, we placed it first, and have also included an additional cross listing of the words by subject groupings.

Index

In Appendix E, at the end of the manual, there is a comprehensive index which lists the major topics and terms of the manual once again, but this time alphabetically.

Conventions Used

Several standard conventions are used to simplify the descriptions. All commands which you are to type in are printed in upper-case type. All 'system' responses are shown as they appear on the screen. 'Control character' entries are denoted by CONTROL-X, where X would be replaced with the actual character entered. Control character entries are made by holding down the CONTROL key while depressing the indicated key.

Request for Feedback

Let us know what you liked and didn't like about this manual. We have tried to make it as complete and friendly as possible, but we know that something, somewhere may be confusing. Let us know if we omitted a useful tip, or explained something poorly. Also, let us know what worked for you so we can continue to produce high quality manuals for future products.

Start-up Procedures

Product Information Card and Replacement Policy

The warranty of this diskette is covered in general by the statement at the bottom of the warranty and disclaimer page in the front of the manual. Since its message is hidden in legalese, let's just say that roughly what is meant is that we did our best to ship the diskette in perfect condition, but we have no control over what happens to it enroute to your disk drive. If, for some reason, it will not 'boot' (come up on the screen when the machine is turned on), then you should take or send it back to the place where you purchased it. If they cannot get it to boot, then we will replace it at no additional cost to you, for a period of 30 days after you purchased it. (Thereafter, a nominal replacement fee may be charged.) Once you have a disk that boots and runs, then it is your responsibility to protect it by using it only for the purpose of making duplicate work disks and backups (see next section).

In the meantime, we would appreciate it if you would fill out the Product Information Card. This card gives us valuable information about our customers and helps us design our products and product line to better serve you. If everyone who buys GraFORTH turns out to be retired and living in Florida, then this manual will have to be rewritten with a different set of jokes. The card also allows us to keep you up to date. If we decide to send out an updated GraFORTH diskette, then you would probably want to know about that.

Making and Using Backup Copies

If you have not yet made a backup copy of the GraFORTH diskette, then now is a good time to do so. Never use the original as a work disk, not even for a few minutes. Particularly, never use an original disk to try to solve a problem which blew up your work disk. Make a new backup if you can, and use that to experiment. Because GraFORTH is compatible with DOS 3.3, any copy program you normally use to copy your 16-sector Apple DOS disks will work to copy this diskette. The COPYA program which came with your DOS 3.3 System Master diskette is a particularly reliable one, and we recommend using it. In fact, it is recommended that you have two backup copies so that if one goes down, you won't have to open your lead-lined vault to get at the original.

A PLAYful Review

An Introductory Tutorial

We suggest that you study the Table of Contents and the Manual Diagram for a few minutes to get an idea of where we are and where we have to go, and then, because we know you are itching to get your hands back onto that machine and create a few three-dimensional forms to rotate in free-floating and free-wheeling space, we'll give you a preview of what's to come in future chapters...

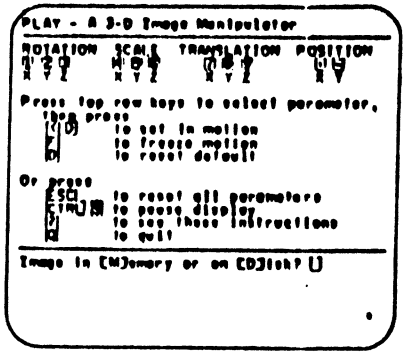
If you catalog your disk, you'll find the text file PLAY on it. PLAY is a set of routines (or "words"), which when compiled and run, allows you to pull up a three-dimensional form off the disk (several are provided), and play with it in 3-D space. Later on, we'll tell you how to use PLAY to understand better the 3-D images you are creating. But for now, we are just going to have some fun using PLAY. If you have not yet made a backup copy of your disk, we'll just have to insist that you do so now. From now on, when we speak of your GraFORTH diskette, we will actually be referring to the copy you use as a work diskette.

Running the PLAY Program

To run PLAY, boot your disk and respond with an 'N' or 'NO' to the demonstration question. When the "Ready" prompt comes on, type

```
READ " PLAY " <return>
```

Be sure to type it exactly as you see it, including the spaces between PLAY and the quotation marks. The word READ is a command that tells the GraFORTH system to read a file on the disk and compile it into the word library, that is, turn it into machine language for the machine to use. When the "Ready" prompt reappears, type 'RUN' and a set of instructions will be displayed on the screen, as illustrated in diagram below:



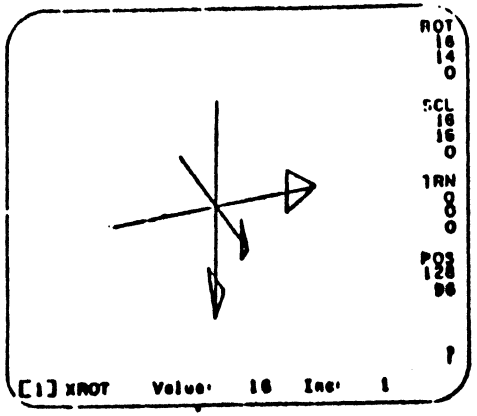
The words, ROT, SCALE, TRANS, and POS refer to the four parameters you may use to manipulate the image in space. ROT stands for the ability to rotate the object around any of three axes, SCALE stands for the ability to change the scale or size of the object, TRANS stands for the ability to translate or move the image in its 'space envelope', and POS stands for the ability to move the position of the image on the screen.

The characters, 123 456 789 :-, are pressed to activate any of the above parameters along any of the axes, X, Y, and Z, indicated below them. The commands in the middle of the screen start and stop the selected action, or reset the parameters to their starting positions (called 'defaults'). You are to press just those keys which are high-lighted in inverse. If the action ever gets too fast for you or you see something you'd like to study, pressing CONTROL-S will stop the action until you press another key. Similarly, 'D' will reset the currently active parameter to its default position, <ESC> will put you back at the beginning, and 'Q' will put you out in the cold at the "Ready" prompt.

At the bottom of the screen, you are being asked to answer a question as to where the image is which you would like manipulated. The quickest way to understand the program is to dive in and try it, pressing the various keys along the way to see their effects. But first, we need an image to play with.

Unless you're way ahead of us, you do not have a 3-D image in memory yet, so select 'D' to answer the question at the bottom of the screen and to begin the image loading process. Next, hit <return> to default the address to 2816 (more on that later), and enter 'XYZ' as the image filename. Again, hit <return>. Your screen should now show a picture of a vertical line crossed by a horizontal arrow. In a moment you'll see that these are really three intersecting arrows. On the right side of the screen are the movement commands, ROT, SCALE, TRANS, and POS. Ignore the latter two for the purposes of this short trial run. Now the fun begins. Press '2', and then the right arrow key. Next press '1', then the right arrow key. Observe the numbers changing over on the right. See if you can figure out what they do as you select keys to press from the previous diagram. Try the left arrow keys, and watch the action and the numbers change. You may freeze the selection last changed with 'F', and also by using the arrow keys to get the parameters back to zero.

At this point, you should have a screen which looks something like the one on the next page.



Now press 'D' to reset your current parameter to its default; get the idea? The more you press the arrow keys, the faster the image will turn. If you are working on a color screen, you will see that each axis is a different color, which may help to keep them straight. Remember, pressing <esc> will set all parameters to their starting (default) positions, which may be needed if they start getting out of hand. In particular, if SCALE, TRANS, and POS get beyond a certain size, they will no longer fit on the screen, and they will begin to "wrap around", appearing quite unexpectedly on the opposite side of the screen. It will look as if you have lines bouncing off the walls, but it is really only wraparound. If you like that effect, then fine; but if not, just keep the numbers smaller.

That's enough fun. We have to get back to work and learn the rest of what GraFORTH has to offer. We'll come back to PLAY in Chapter 8, and learn what TRANS and POS actually do. But if you just can't quite quit yet, we'll mention (while the boss is out of the room) that the way to bring up another 3-D image to PLAY with is to type 'Q' and then RUN again, repeating all steps except the one where you enter the filename (try HOUSE).

CHAPTER TWO: BACKGROUND

CHAPTER TABLE OF CONTENTS:

What You'll Need to Have

Hardware Requirements

GraFORTH requires that you have the following minimum hardware components:

An Apple or Apple + computer with 48K RAM
One DOS 3.3 Apple disk drive with controller
A black and white (or green) video monitor, and/or
A color monitor or color TV with an RF modulator

Recommended Peripheral Options

In addition to the above (including the color display), it is highly recommended that you have a 16K RAM or language card, to provide more available memory, and a second disk drive.

Software Requirements

GraFORTH is written in 6502 machine language using the ALD SYSTEM Assembler which was written by Paul Lutus and is also available from Insoft. All graphics are internal and are therefore completely independent of either Apple BASIC (INTEGER or APPLESOFT). GraFORTH boots from the 'monitor', without a BASIC 'HELLO' program, as you will notice by the presence of the asterisk prompt (rather than the BASIC prompt), during bootup. This makes the boot program independent of any resident language in ROM, avoiding the differences between Apple II and Apple II+ machines which are sometimes troublesome to software. It also means, however, that it is not possible to add your own special HELLO program to the disk to have it do your favorite tricks on bootup. But don't despair; we will show you later how to have GraFORTH automatically run any program you wish on bootup. Further, that program can be written directly in GraFORTH.

What You'll Need to Know

What You'll Need to Know about Your Machine

While it is intended that this manual serve as a tutorial in the use of the GraFORTH language, it is not intended to cover material already covered quite thoroughly and thoughtfully in the set of manuals distributed by the Apple Computer Company. If you are a new user, unfamiliar with how to use your Apple computer, we suggest that you take the time to go through the Apple Reference Manual, which came with your machine. You will not need to know everything in it to use your Apple successfully, but the more you know, the easier it will be to understand operations which might otherwise seem puzzling.

What You'll Need to Know about the DOS

With the exception of certain small changes (see below), GraFORTH uses the standard Apple Disk Operating System, Version 3.3, known affectionately as DOS 3.3. If you are at all unfamiliar with how to use your disk operating system, we suggest you take the time now to study the DOS Manual which came with your disk drive(s). It will be time well spent.

Minor Modifications In DOS 3.3

Minor modifications have been made in the disk operating system to make it run smoothly with GraFORTH. Most of these changes will be 'user-transparent', or not noticeable, and using DOS from GraFORTH is the same as using DOS from either of Apple's BASICs. Both create TEXT type data files, and GraFORTH even uses TEXT files for saving program 'source code'. The DOS on the supplied diskette has been modified, however, to take advantage of an existing language card or RAM card. If you have such a card, DOS will be loaded automatically into the language card, leaving much more room (almost 10K) in main memory for program development. To take advantage of this additional memory, two editors have been provided on the disk; OBJ.EDITOR1 for systems without language cards and OBJ.EDITOR2 for systems with language cards. Note that GraFORTH requires the DOS it is supplied with. You can not transfer GraFORTH to a disk with a different DOS!

Making Space on the Disk

The GraFORTH diskette, as delivered, is nearly full. Not only does the disk contain all the system files needed to use GraFORTH, it also contains many demonstration files as well as some specialty files. After you have copied the diskette and exhausted your interest in the demos, you may want to trim your work disk down a bit to make room for your own files. The demonstration programs will probably be the first to go. Appendix C lists the files on the disk, indicating those which may be deleted without danger to the GraFORTH system by a ">". See the sections which follow for help on how to delete files from your work disk.

Alternatively, you might want to leave your work disk intact and set up another disk for program development. The GraFORTH system would not need to be on such a disk; you could use, instead, a standard DOS diskette. If so, you will need to copy the editor file or files onto that disk as the GraFORTH word, EDIT, looks for the editor program on the 'current drive'. If you are using a language card, copy OBJ.EDITOR2 onto your program development disk, otherwise copy OBJ.EDITOR1 onto that disk.

Deleting Files

There are three simple ways to delete files from the disk. One way is to boot an Applesoft disk, then catalog the GraFORTH diskette and delete the files you want to remove as you would on a standard DOS disk. Alternatively, you could use your favorite file utility, such as FID on your DOS 3.3 System Master Disk, or else boot GraFORTH and enter your DOS commands from the program itself. If you are already in GraFORTH, the latter method is the method of choice. To delete files directly from the program, you will need to take the following steps:

1. Boot GraFORTH and you will see the prompt

   Demonstration (Y/N)?

2. Answer 'N' and the "Ready" prompt will appear.

3. Respond with:

   EDIT <return>

The drive will whirl a bit, loading the editor, and then the editing title will appear along with a flashing cursor.

4. To enter a DOS command, type:

   ConTRoL-D <return>

and the following prompt will appear:

   Enter DOS Command :

5. Respond with:

   CATALOG <return>

   (or CATALOG,D1 <return> for two drive systems)

and the catalog will be listed.

6. Select the files to be deleted and type:

   DELETE filename <return>

The drive will run briefly, make its usual scratching sounds and the file's name will be deleted from the disk directory. You may confirm that fact with another CATALOG command. Then repeat the procedure to delete the other files you wish to remove from the disk. To return to the editor, press the <return> key twice without entering any DOS commands, and you will see the blinking cursor of the editor once again. To return back to GraFORTH, type 'BYE', then press <return>. The GraFORTH header and the "Ready" prompt should reappear.

Entering Other DOS Commands

The above steps represent the procedure to be followed to enter any standard DOS 3.3 command from GraFORTH itself. Later on, we'll describe another method which enables you to use DOS commands from the "Ready" prompt directly without entering the editor.

What You'll Need to Know about Disk Care

We assume that by now you have made a copy of the original diskette, have stored it in some safe place, have had some fun with PLAY and are anxious to get down to "work". Bear with us for one more cautionary remark (admittedly unnecessary for almost all of you). In case you are not familiar with the care and feeding of floppy diskettes, what we mean by "safe place" is that the disk is stored vertically, is not bent or folded or exposed to magnetic fields or to temperatures outside of the range 50 to 125 degrees F., and that the "naked" portion of the disk (as seen through the small oval opening in the plastic covering) is not exposed to dust, fingerprints, or cigarette ashes. We recommend that you always keep your disk in its protective sleeve and box whenever it is not actually in a disk drive. Never attempt to write on it with a pencil or ball-point pen. If treated in this way, your diskettes should give you years of devoted service, and perhaps even become collector's items of considerable value to your grandchildren (well, at least curiosities).

What You'll Need to Know about Programming

It is not necessary to know how to program to learn programming in GraFORTH. It is our position that both TransFORTH and GraFORTH are simple enough to learn that novices can take them on as beginning languages. We also believe that they are so powerful that advanced programmers can use them in a full range of commercial applications. While it is not necessary to learn programming prior to starting in on GraFORTH, if you are already familiar with BASIC or another high-level language, you will, of course, learn GraFORTH much faster. In particular, a familiarity with Applesoft and/or Apple Pascal will speed the learning of the control structures, data structures, and the file handling portions of the language. Familiarity with FORTH will give you a head start on the operation of the stacks, postfix notation, and the word library.

What You'll Need to Know about Graphics

Here again, prior experience in graphics programming is helpful to learn programming in GraFORTH, but it is not required. Graphics is the heart of GraFORTH - all kinds of graphics - standard two-dimensional graphics, TURTLEGRAPHICS, color graphics, block printing of image fonts, three-dimensional graphics, all at speeds which will support animation, and set to music if you like. If you do not intend to do a lot of graphics programming with GraFORTH, then you may have the wrong language. (Perhaps you really need TransFORTH...)

With GraFORTH, powerful graphics editors allow your images to be created with considerable ease. A powerful command set allows them to be put in motion. Routines can be set up as independent words, then tested out and stored, to be used again and again. But you do not need to know it all before you start. We'll take you through it a step at a time.

However, if you are a beginner at graphics, you will learn faster if you draw upon several sources at once and approach the subject from all sides. The Applesoft Tutorial has a good introduction to Apple graphics, as does the Apple User's Guide by Lon Poole, et al. The Apple Pascal Language Reference Manual has a good chapter on TURTLEGRAPHICS, and if you really want to get into the whole subject, try Graphic Software for Microcomputers by B. J. Korites (Kern Publications, 1981).

What You'll Need to Know about Music

As mentioned above, one of the features of GraFORTH is a music synthesizer which enables you to add music to the programs you write in the language. Operation is straightforward, and a note table is provided to make use of the music synthesizer as simple as possible. We think you will be amazed at the added dimension it will give to your programs.

What You'll Need to Do

Get an Overview

One of the most time-saving things you can do right now is to get an overview of the manual and the structure of GraFORTH. Time spent on the demos, and studying the table of contents and diagrams will give you a general framework which then just needs to be filled in with detail.

In between this chapter and Chapter 9 are the chapters which explain in detail how to use GraFORTH. Chapter 3 gives an introduction to the use of GraFORTH. It is something of a mini-manual in itself, and even those of you who know FORTH may find it a useful review of how GraFORTH differs from other FORTH languages. The next six chapters build somewhat on one another and should be taken in order, with the possible exception of Chapter 9 on music, which could be read and used anytime after Chapter 5.

Run the Demos

The set of demo programs on the diskette will give you a good sense of what GraFORTH can do. To run a demo, just answer 'Y' to the demo question which appears after bootup, and then simply select from the menus which follow. Later we shall tell you how to remove the demo question.

Plunge In

At this point, there is very little left to do but to load your work copy of GraFORTH in the drive, boot it up, and plunge in. Start at a place in the manual appropriate for your skills and knowledge, read that section, turn to the program, work the examples, and then see if you can amaze yourself with a few examples of your own. That's all there is to it. Remember, the chapters, like the language, tend to build sequentially, so it may not be wise to skip around too much.

What You'll Need to Be

Confident, fearless, and fun-loving. Willing to take risks, make mistakes, and learn from those mistakes. Willing to ask stupid questions and make a fool of yourself to find out what you need to know. Willing to let yourself enjoy life and turn work into play. In short, just your average, run-of-the-mill, Apple owner.

CHAPTER THREE: STARTING GraFORTH

Chapter Table of Contents:

Purpose and Overview

As you'll soon see, GraFORTH is a complete, structured language, with all of the interesting nuances of such a language. In this chapter, we'll introduce GraFORTH as a language. We'll discuss the GraFORTH system, the word library (sometimes called the dictionary), and the concept of 'words'. We'll show you how to use the stack to do arithmetic using Reverse Polish Notation, and then define your own words in terms of existing ones. We'll discuss the looping and control features of GraFORTH, then tie up the chapter with some rules of thumb for writing programs in GraFORTH.

This chapter (as well as the others) contains numerous examples to help you understand the GraFORTH system. We strongly encourage you to try these examples on your computer. And as you gain experience with the concepts, we encourage you to experiment further, so that you become truly comfortable working with GraFORTH.

First Things First

Insert your GraFORTH disk in the drive and boot it. After a few seconds you'll see:

```
GraFORTH ][ (C) P. Lutus 1981

Demonstration (Y/N) ?
```

If you haven't yet seen the GraFORTH system demonstration, you might want to do that now. The demonstration includes explanations of what GraFORTH is and what it does. As we go on, however, we'll ignore this question, assuming that you've either already seen the demo or are no longer interested. Later, we'll show you how to remove the demo question entirely... Now let's get into the language. Type an 'N' to the demonstration prompt, and you will see:

```
GraFORTH ][ (C) P. Lutus 1981

Ready
```


The word "Ready" appears whenever the system is ready for your input. (Makes sense...) If at any time you do not see the word "Ready" when you think you're supposed to, then it may be time to start wondering... With the word "Ready" beckoning you on, let's back up for a few moments to discuss GraFORTH.

The System

The language can be divided into two main parts. The first part contains the compiler and low-level system routines. For most applications, the internal workings of these routines can be ignored. They usually do the things which need to be done without a lot of fanfare. The second part of the system is the 'word library'. The word library is the "visible" part of the GraFORTH system, and is the basis for writing programs.

Words

The word library is made up of a large number of GraFORTH 'words'. You can see this list of words at any time by typing the word "LIST". LIST is a GraFORTH word that lists all of the GraFORTH words. (LIST will display 20 words at a time. To see the entire list, press <return> at each pause. Press CONTROL-C if you want to stop the listing.)

Each GraFORTH word accomplishes a particular task. For example, the word "BELL" beeps the Apple speaker, the word "+" adds two numbers together, and the word "DRAW" draws a three-dimensional image on the screen. Nearly everything in GraFORTH is either a word or a number. Words can be programs, subroutines, variables, or strings. Programs are written, not by entering "program lines", but by stringing words together.

The name of a word can be any string of ASCII characters that does not include a space or carriage return. The space acts as a divider between words, and a carriage return tells the system to compile the entered line into machine language and, in most cases, execute it. Since GraFORTH uses spaces to determine when one word ends and another begins, putting spaces between GraFORTH words is very important.

The Data Stack

Words are executed in the order they are entered. When the word "+" is executed, it wants to add two numbers together, right then and there. This means that both of the numbers to be added must already be available for "+" when it is executed. Where do the numbers wait before they are added? They are on the 'data stack', placed there by you before entering "+".

All numbers in GraFORTH are routed through the data stack, which we'll usually just call the 'stack'. The stack is simply a stack of numbers, one on top of another, much like a deck of cards, or a stack of dinner plates. When you enter a number, it is put on the top of the stack, above any numbers which might already be there. Some words place numbers on the stack. Some words remove numbers from the stack. Some words do both. The word "+" is an example of this; it removes two numbers from the stack, adds them, and places the sum back on the stack. If the stack is empty, and a word tries to remove a number from the stack, a phenomenon called 'stack underflow' occurs. Stack underflow will be discussed in greater detail at the end of this chapter.

Numbers

GraFORTH is an integer language. It uses numbers in the range -32768 to +32767. You can enter numbers outside of this range, but they will be "folded" back into the range (e.g. the number 32769 will be stored as -32767). Certain operations, such as division, will truncate decimal numbers back into integers. For example, 7/3=2.333333, but GraFORTH will evaluate 7/3 as 2.

Hands-On Experience

Nearly every entry in GraFORTH is ended by pressing the <return> key. For the examples below, and throughout the rest of the manual, press the <return> key after every entry unless we tell you otherwise.

As you step through these examples, you may mistype something, and find yourself in a situation you don't quite yet know how to get out of. If you can't recover things properly, don't worry: The power switch was put on the Apple for a good reason! Just turn the power off and reboot again, then try to figure out what went wrong. We'll help you along the way.

Enough theory. Let's try some examples. Type:

```
Ready 3 4 5
```

The numbers 3, 4, and 5 have been put onto the stack. If you have any doubts, just type the word STACK.

```
Ready STACK

[3]
[4]
[5]
Ready
```

Typing STACK turns on the stack display, so you can see what numbers are on the stack. The stack display stays on until you type STACK again. This display is toggled on or off whenever you type STACK. You may want to try this a bit, but as we go on, have the stack display on. Now type:

```
Ready 6 7

[3]
[4]
[5]
[6]
[7]
Ready
```

The numbers 6 and 7 have been added to the top of the stack. Notice that the stack display is "upside-down": What we've been calling 'top of stack' is shown as being below the other numbers. Here's why: stacks and 'top of stack' are both standard computerese conventions, and we didn't want to break tradition by calling it the "bottom of stack". But the GraFORTH stack can hold up to 128 numbers while the Apple screen can only display 24 lines. With the stack display turned upside-down, then the 'top of stack' (the most accessible number) will always be the number closest to the "Ready" prompt, instead of being scrolled off the screen.

Now that we have some numbers on the stack, what can we do with t[illegible] One thing we can do is print them. The word "." (period) removes a number from the stack and prints it. Type a period.

```
Ready .
7
[3]
[4]
[5]
[6]
Ready
```

The 7 was removed from the stack and printed. Now type "+":

```
Ready +

[3]
[4]
[11]
Ready
```

The numbers 5 and 6 were removed from the stack by the word "+", added together, and the sum placed back on the stack. Now type three periods, separated by spaces:

```
Ready . . .
1143

Ready
```

The 11, 4, and 3 were all printed, without any spaces between them. We'll show you how to position the printing of both numbers and text in a bit.

You now know how to put numbers on the stack, add them together, and remove them by printing them. Since most words in GraFORTH use the stack, it's important to know exactly what's happening on the stack when a word is executed. Let's introduce a notation for the effect of a word on the stack. We'll list the word, followed by a "before and after" representation of the stack, then a brief description of what the word does. The stack numbers are shown as letters, with a dash to the right indicating top of stack. Remember, the top of stack is the dash on the right. An empty stack is indicated by three dashes. Using this notation, here are the four GraFORTH words we've shown so far:

Word	Before	After	Description
LIST	- - -	- - -	Lists the words in the GraFORTH word library.
STACK	- - -	- - -	Toggles the stack display on and off.
.	n -	- - -	Prints n.
+	m n -	p -	Takes m and n off the stack, adds them and places their sum, p, back on the stack (p=m+n).

Note that there may be other numbers on the stack below those shown in the before and after diagrams, but these are not affected by the word.

More Words

Stack Words

Here are some GraFORTH words which manipulate the numbers on the stack:

DUP duplicates (makes a copy of) the top number on the stack.

SWAP swaps the position of the top two stack entries.

DROP removes the top number from the stack. The number is lost.

OVER makes a copy of the number immediately beneath top of stack, placing the copy on the top of the stack.

PICK uses the top number on the stack to select a number from within the stack, then the number is copied to top of stack. For example, 1 PICK is equivalent to DUP, and 2 PICK is equivalent to OVER.

Here are the same words defined using the stack diagram:

Word	Before	After	Description
DUP	n -	n n -	Duplicates n.
SWAP	m n -	n m -	Swaps m and n.
DROP	n -	- - -	Drops (forgets) n.
OVER	m n -	m n m -	Copies m to top of stack.
PICK	...m n -	...m q -	Copies nth item to top of stack.

Keeping an eye on these definitions, some more examples may be helpful here:

```
Ready 1 2 3
[1]
[2]
[3]
Ready SWAP          (Exchange positions of the 2 and 3.)
[1]
[3]
[2]
Ready DUP           (Make a copy of the 2.)
[1]
[3]
[2]
[2]
Ready DROP          (Remove the copy just made.)
[1]
[3]
[2]
Ready OVER          (Copy the second from top of stack.)
[1]
[3]
[2]
[3]
```



```
Ready 4 PICK        (Copy the fourth position down stack.)
[1]
[3]
[2]
[3]
[1]

Ready DROP DROP .   (Remove 3 and 1, then print the 2.)
2
[3]
[1]
Ready DROP DROP     (Remove the remaining 3 and 1.)

Ready               (The stack is now empty.)
```

You will probably want to experiment further with each of these words with the stack display on. While their functions may not be terribly exciting, you'll find they will be very useful later on for placing numbers where they need to be at the right time.

Arithmetic Words

You've seen how "+" works; on the next page is a listing of the GraFORTH arithmetic words.

Word	Before	After	Description	
+	m n -	p -	p=m+n	(addition)
-	m n -	p -	p=m-n	(subtraction)
*	m n -	p -	p=m*n	(multiplication)
/	m n -	p -	p=m/n	(division)
MOD	m n -	r -	remainder	(modulo)
CHS	n -	m -	m=-n	(change sign)
ABS	n -	m -	m=ABS(n)	(absolute value)
SGN	n -	m -	m=1 if n>0, 0 if n=0, -1 if n<0	(sign)
SIN	n -	m -	-128<m<127	(sine)
MIN	m n -	p -	p=m if m<n, n if n<m	(minimum)
MAX	m n -	p -	p=m if m>n, n if n>m	(maximum)
RND	- - -	n -	-32768<n<32767	(random number)
RNDB	- - -	n -	0<n<255	(random byte)

Here are some examples of the GraFORTH arithmetic words in action:

```
Ready 23 5 / .
4
Ready 23 5 MOD .
3
```

(23 divided by 5 leaves 4, and a remainder of 3.)

```
Ready 6 CHS
[-6]
Ready ABS .
6

Ready 18 19 MIN
[18]
Ready SGN .
1

Ready -7 SGN .
-1

Ready RND .
-22317
```

RND leaves a random number on the stack. (Of course, the number displayed will most likely be different from the one shown above.)

Using Words

Now that we've introduced a whole slurry of words, let's put them to use.

For these examples, we'll assume the stack is empty before beginning. There are a few ways to empty the stack. With the stack display on, you can type either DROP or "." repeatedly until the stack display shows the stack is empty.

Another way to clear everything is to type the word ABORT. ABORT restarts GraFORTH, resetting things back to their initial conditions. ABORT can be handy when used from the keyboard, but if executed from a running program, it stops the program immediately. (There is an exception to this which will be discussed in Chapter 5.)

As you've already seen, the way to add two numbers is to enter the numbers first, then type "+".

```
Ready 3 4 + .
7
Ready
```

This notation, where the numbers precede the operator, is called Postfix, or Reverse Polish Notation, and is used in all versions of Forth, as well as in most Hewlett-Packard calculators. Its main advantage over "standard" notation is that complicated expressions can be evaluated without having to use parentheses. For example, if you wanted to add 3 and 5 together, add 7 and 9 together, then multiply their sums in a language like Basic, you would type:

```
X=(3+5)*(7+9)
```

Note that since Basic always multiplies before adding, parentheses were needed to group the sums together. In GraFORTH, you can solve the problem this way:

```
Ready 3 5

[3]
[5]
Ready +

[8]
Ready 7 9

[8]
[7]
[9]
Ready +

[8]
[16]
Ready *

[128]
Ready .
128
Ready
```

This example was "unfolded" so you can see exactly what is happening on the stack. Usually, the entire expression is entered on one line:


```
Ready 3 5 + 7 9 + * .
128
Ready
```

To find the cube of a number, you can type the number three times and multiply:

```
Ready 3 3 3 * * .
27
```

Another way is to type the number once and use DUP to duplicate it:

```
Ready 3 DUP DUP * * .
27
```

DUP allows you to use any number without having to enter it repeatedly. This will be very useful for general purpose operations inside programs.

Printing Text

Printing text in GraFORTH is straightforward: type the word PRINT, the word " (quote), the text to be printed, then another quote:

```
Ready PRINT " SUPER ZAPPO SPACE GAME "
SUPER ZAPPO SPACE GAME
Ready
```

Since the quote is a GraFORTH word, the spaces between the quotes and the text are required. Note that you can use quotes within the quoted text, as long as it is not separated on both sides with spaces:

```
Ready PRINT " THIS IS THE "BEST" GAME EVER! "
THIS IS THE "BEST" GAME EVER!
Ready
```

Since PRINT does not automatically print a space or a carriage return at the end of the text, two other handy words to know are SPCE and CR. SPCE prints a space, and CR issues a carriage return. Notice the difference in the following three examples:

```
Ready PRINT " FIRE " PRINT " ONE "
FIREONE

Ready PRINT " FIRE " SPCE PRINT " TWO "
FIRE TWO

Ready PRINT " FIRE " CR PRINT " THREE "
FIRE
THREE
```

Printing text is not very useful if the system only prints the text immediately then forgets it. Fortunately, GraFORTH can do much more than that.

Defining New Words

The power of GraFORTH as a language lies in the ability to define new words in terms of old ones. In fact, writing "programs" in GraFORTH is done by simply defining a series of new words which accomplish the desired task. These new words are added to the word library and can be seen by typing the word LIST. In this way, the GraFORTH language itself (of which the word library is a part) "expands" to become your program!

New words are created with 'colon definitions' (so named because they begin with a colon). The form for a colon definition is:

: <word name> <string of defining words> ;

The colon tells the system to begin a new word definition. The name that immediately follows the colon will be the name of the new word. The words that follow the name make up the "definition" of the word; they are the words to be executed whenever the defined word is typed. These words behave just as if they had been typed in directly at the keyboard. The semicolon marks the end of the colon definition, and causes the word to be compiled into machine language and added to the word library.

As an example, let's define a word that adds two numbers then prints their sum along with a short message:

```
Ready : SUM PRINT " THE SUM IS  " + . ;
```


Following the form for colon definitions, SUM is the name of the new word, and

PRINT " THE SUM IS " + .

is executed whenever the word SUM is entered. The word PRINT causes the phrase "THE SUM IS" to be printed, the + adds the top two numbers on the stack, and the period prints the sum. (Note that there are two spaces between the word IS and the quote, so that a space will appear between the text and the number.) Now let's try our new word:

```
Ready 25 31 SUM
THE SUM IS 56
Ready
```

LIST the word library, and you'll see that the word SUM has been added:

```
Ready LIST

SUM
CHS
SGN
CALL
:
.
```

A nice addition to this word would be to reprint the numbers being added. But before we commit ourselves to a colon definition, let's try it "live", where we can watch things one step at a time:

```
Ready STACK

Ready 25 31

[25]
[31]
```

We need to make copies of the two numbers: one set will be reprinted on the screen, and other set will be added together. (Remember that many GraFORTH words consume numbers from the stack, so we need to have the numbers ready to "feed" them!) The quickest way to copy a pair of numbers is by using OVER OVER:

```
Ready OVER .

[25]
[31]
[25]
Ready OVER

[25]
[31]
[25]
[31]
```

Now let's reprint the first set of numbers along with some informative text:

```
Ready PRINT " THE SUM OF  " .
THE SUM OF 31
[25]
[31]
[25]

Ready PRINT "  AND  " . PRINT "  IS  "
 AND 25 IS
[25]
[31]
```

Now let's add the numbers...

```
Ready +
[56]
```

...and print the sum:

```
Ready .
56
```

Now let's put it into a colon definition, with a different name. Note that you can enter the definition over several lines (if you like).

```
Ready : SUM1

Ready OVER OVER PRINT " THE SUM OF  " .

Ready PRINT "  AND  " .

Ready PRINT "  IS  " + . ;
```


After entering the definition, the word SUM1 is also on the word library:

```
Ready LIST

SUM1
SUM
CHS
ABS
.
.

Ready 25 31 SUM1
THE SUM OF 31 AND 25 IS 56
```

SUM1 can now be called at any time, from either the keyboard or another word definition, as easily as any of the original GraFORTH words in the word library.

Note: As you write and enter colon definitions, be sure to enter a semicolon to finish the definition! If you don't, GraFORTH will assume that everything you type is part of a word to be executed at a later time. If GraFORTH ever responds to words like LIST with only a "Ready" prompt, you've probably left a semicolon out of colon definition.

Forgetting Words

You can see that if we keep on defining new words, the word library will continue to grow until we use up all of the memory available. Sometimes words are no longer needed, or a word might contain a mistake (???). In either case, to delete one or more words, the word FORGET is used. It takes the form:

```
Ready FORGET <wordname>
```

FORGET cannot selectively remove words from the middle of the word library. It only truncates off the top, deleting the specified word and every word above it. In our example, to delete both SUM and SUM1, type:

```
Ready FORGET SUM
```

```
Ready LIST

CHS
ABS
SGN
.
.
.
```

Notice that both SUM and SUM1 were removed from the word library. Had there been more words above them, they would also have been removed.

Note: You will not get an error message if you try to FORGET a word that is not in the word library. This makes implementing program 'overlays' easier. (Overlays will be discussed in Chapter 5.) However, if you misspell the word you want to forget, then no words will be deleted from the word library. Thus, it's a good idea to use LIST to verify that the right word or words have been deleted.

Looping Structures

The GraFORTH DO - LOOP construct is available for repetitive tasks where the number of repetitions is known ahead of time. The form for a DO - LOOP is:

<ending value> <initial value> DO <words to be repeated> LOOP

The word DO removes two values from the stack. The top number is used as an 'initial value' and the next number is used as an 'ending value'. The words between DO and LOOP are executed, then the initial value is incremented by one. If this incremented value (which we'll call the 'loop value') is still less than the ending value, the program loops back to execute the words between DO and LOOP again. This cycle is repeated as long as the loop value is less than the ending value.

If you are familiar with Applesoft Basic, you will notice that DO - LOOP is similiar to Applesoft's "FOR -- NEXT" looping structure.

It is often handy to retrieve the current loop value. Inside the DO - LOOP, the word "I" retrieves the loop value and places it on the stack. Here is an example:

```
Ready 5 0 DO PRINT " HERE IS NUMBER  " I . CR LOOP
HERE IS NUMBER 0
HERE IS NUMBER 1
HERE IS NUMBER 2
HERE IS NUMBER 3
HERE IS NUMBER 4
```

"5 0 DO" sets up the looping structure for 5 loops. Inside the loop, the phrase "HERE IS NUMBER" is printed, then the loop value is retrieved by I, then printed with ".". CR causes the carriage return to put each number on its own line, and LOOP marks the end of the loop, causing the loop value to be incremented and compared with the ending value. Note that the loop continues only as long as the loop value is less than the ending value. That's why the loop stops at 4, not 5 as in Applesoft.

The words DO and LOOP work as a pair and must always be matched up, either on the same line together or entered in a colon definition. Typing DO or LOOP alone can have nasty and unpredictable results.

To make a loop with an increment other than 1, use +LOOP instead of LOOP. +LOOP removes a number from the stack to use as the increment. This number can be either positive or negative (for loops that count backwards). Here is an example:

```
Ready 10 0 DO I . CR 2 +LOOP
0
2
4
6
8
```

The 2 was used by +LOOP as the increment.

```
Ready 150 200 DO I . CR -10 +LOOP
200
190
180
170
160
```

Loops can be nested inside one another. The loop value for the current innermost loop is always accessed by "I", and the loop value for the next outer level is accessed with the word "J", as in this colon definition:

```
Ready : DOUBLELOOP

Ready 4 0 DO

Ready    PRINT " OUTER LOOP:  " I . CR

Ready    3 0 DO

Ready       J . SPCE I . CR

Ready    LOOP

Ready LOOP ;
```



```
Ready DOUBLELOOP
OUTER LOOP: 0
0 0
0 1
0 2
OUTER LOOP: 1
1 0
1 1
1 2
OUTER LOOP: 2
2 0
2 1
2 2
OUTER LOOP: 3
3 0
3 1
3 2
```

The inner loop is cycled three times for each cycle of the outer loop. Note that the outer loop value is referenced in the outer loop with "I", but is referenced from the inner loop with "J". Just remember that "I" always references the loop value for the current innermost loop.

If more than two nested loops are being used, the loop value of the third loop out can be accessed from inside the innermost loop with the word "K".

The Return Stack

DO - LOOPs make use of another stack in the GraFORTH system, similar to the data stack, known as the 'return stack'. The return stack can also hold 128 numbers, though for most programs it rarely contains more than a few. (Most versions of Forth, because they are interpreted, use the return stack for a variety of purposes. Because GraFORTH is compiled directly into machine language, the Apple's processor itself takes care of these things.)

When the word DO is encountered, the top two values on the data stack are moved over to the return stack, with the loop value on top and the ending value underneath. The word LOOP increments the loop value on the return stack. The word "I" places a copy of the top return stack value and places it on the data stack. When the loop is finally exited, the two return stack values are removed.

There are a few words in GraFORTH that enable you to use the return stack directly. The return stack can be a handy place to put numbers for a moment while playing games with other numbers on the data stack. (In Chapter 5 we'll show you how to declare variables for more permanent storage.) Care should be taken to avoid disturbing the value and placement of existing return stack entries when using DO - LOOPs. (In other words, if you're not sure, don't!) Here are the words that directly control the return stack:

PUSH moves the top data stack entry to the return stack.

PULL moves the top return stack entry back to the data stack.

POP removes the top return stack entry. The number is lost.

Suppose there are three numbers on the stack and you want to reverse the order of the bottom two. Here is one way to do it:

```
[3]
[2]
[1]
Ready PUSH

[3]
[2]
Ready SWAP

[2]
[3]
Ready PULL

[2]
[3]
[1]
Ready
```


Comparing Numbers

A number of GraFORTH words are devoted to comparing numbers. These words are:

<>	(not equal to)
=	(equal to)
>	(greater than)
<	(less than)
>=	(greater than or equal to)
<=	(less than or equal to)

Each of these words removes two numbers from the stack, comparing the second stack number down with the top stack number, and returns on the stack either a 1 if the comparison is true, or 0 if the comparison is false. Here are a few examples:

```
Ready 5 5 = .
1

Ready 5 7 = .
0

Ready -32 -6 < .
1

Ready 45 46 >= .
0
```

A couple of other words related to the comparison words are AND and OR. These words remove two numbers from the stack and perform a logical operation between each of the 16 bits of the numbers, returning another number to the stack.

AND performs a bitwise "AND" between the two stack values; OR performs a bitwise "OR". Don't worry if you're unfamiliar with the relationships between numbers and their bits. Usually the importance of AND and OR is between two zero or nonzero numbers:

If both the top stack value and the second stack value are nonzero (representing "true"), then the AND of the two numbers will also be nonzero. If either or both numbers are zero, then the AND will also be zero.

If either the top stack value or the second stack value are nonzero, then the OR of the two numbers will be also nonzero. Only when both numbers are zero will the OR operation be zero.

AND and OR are useful for combining the results of two or more tests. The following example tests whether or not a given number is greater than 5 and less than 10. We'll test with two numbers, 7 and 3:

```
Ready 7
[7]
Ready DUP 5 >
[7]
[1]
Ready SWAP
[1]
[7]
Ready 10 <
[1]
[1]
Ready AND
[1]
```

7 is greater than 5 and less than 10.

```
Ready 13
[13]
Ready DUP 5 >
[13]
[1]
Ready SWAP
[1]
[13]
Ready 10 <
[1]
[0]
Ready AND
[0]
```

13 is not greater than 5 and less than 10.

Decision and Branching Words

An essential part of a computer language is the ability to test a condition, then make a decision on the basis of the test. GraFORTH has five different constructs that accomplish this. Each of the constructs contains a word which removes a number from the stack. In most cases, the "decision" is made on the basis of whether the number is zero or nonzero. Any nonzero number represents a condition being true, and a zero represents false. (Note that the above comparison words place a one on the stack if the comparison is true, and zero if the comparison is false.)

A simple flowchart is included with each of the following constructs, showing the "flow" of the program. The arrows indicate what is executed in what order. The boxes represent a group of words to be executed. The diamonds represent a test, usually for a zero or nonzero number.

Note: Each of these constructs is made up of two or more words. Like DO - LOOP, these decision words work together, and cannot be entered alone. They must be entered either on one line or from within a colon definition.

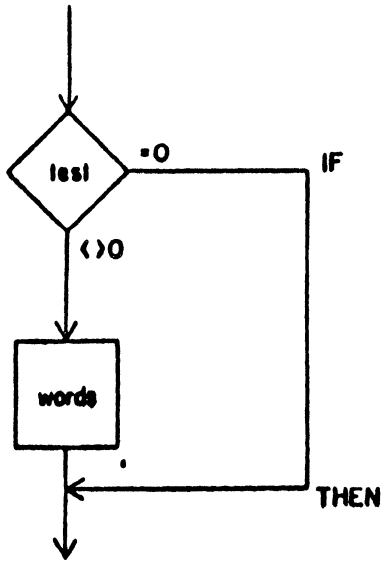
IF - THEN

The simplest decision construct is IF - THEN. The form for IF - THEN is:

```
<stack test value> IF
  <words to be executed>
```

THEN

The word IF removes a number from the stack. If the number is not zero, then the words between IF and THEN are executed. If the number is zero, then the words between IF and THEN are skipped over. In either case, the program continues on after the word THEN. The flowchart for IF - THEN follows on the next page:



Let's use IF and THEN in a couple of colon definitions:

```
Ready : TEST1

Ready PRINT " THE NUMBER IS  "

Ready IF PRINT " NOT  " THEN

Ready PRINT " ZERO. " ;
```

The first and third PRINT words are executed every time. The word IF removes a number from the stack (which we'll supply before we execute TEST1). If the number is nonzero, then PRINT " NOT ", which is sandwiched between the IF and THEN, is executed. If the number is zero, then it is not executed.

```
Ready 5 TEST1
THE NUMBER IS NOT ZERO

Ready 0 TEST1
THE NUMBER IS ZERO
```

IF - THEN constructs can be used with number comparison words. Remember that these words return either one or zero, depending on the success or failure of the comparison. Suppose that for some application, you want to set a limit on the size of numbers. The following word will let any number less than 25 pass through "unharmed", but any number over 25 will be replaced with a 25:


```
Ready : UPPERLIMIT

Ready DUP

Ready 25 > IF

Ready    DROP 25

Ready THEN ;
```

The word DUP makes a copy of the top stack value. The word ">" compares the copy with the number 25, leaving a one on the stack if the number is greater than 25, or a zero if it is not. The word IF removes the one or zero from the stack to decide whether or not to execute the following words. Remember that the original number is still on the stack. If the comparison is false, then the words between IF and THEN are not executed, and the number is left intact. If the comparison is true, then DROP 25 is executed, which removes the original number from the stack and replaces it with 25.

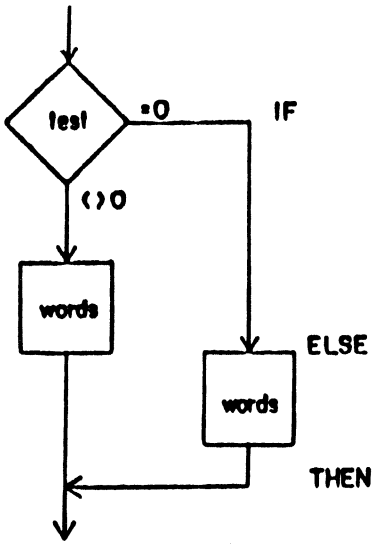
```
Ready 16 UPPERLIMIT .
16
Ready 37 UPPERLIMIT .
25
Ready
```

IF - ELSE - THEN

Another version of the IF - THEN construct is IF - ELSE - THEN. The form is:

```
<test stack value>
IF
   <words executed if nonzero>
ELSE
   <words executed if zero>
THEN
```

As before, the word IF removes a number from the stack. However, if the number is nonzero, then the words between IF and ELSE are executed. If the number is zero, then the words between ELSE and THEN are executed. The program then continues after the word THEN. The flowchart for IF - ELSE - THEN follows on the next page.




```
Ready : TEST2

Ready DUP 100 > IF

Ready    . PRINT "  IS GREATER THAN 100 "

Ready ELSE

Ready    . PRINT "  IS LESS THAN OR EQUAL TO 100 "

Ready THEN ;
```

Again, we've duplicated the number before comparing so that we could print it later, using one of the two periods inside the IF - ELSE - THEN. Also note that the controlled words are indented. This is certainly not a requirement, but it greatly improves the readability of the word definition. (In the next chapter, we'll show you how to use the text editor to save the text of the word definitions.)

```
Ready 106 TEST2
106 IS GREATER THAN 100

Ready 54 TEST2
54 IS LESS THAN OR EQUAL TO 100
Ready
```

As with loops, IF - THEN constructs can be nested. This example puts checks for both upper and lower limits on a number:


```
Ready : TWOLIMITS

Ready  DUP 25 > IF

Ready     PRINT " GREATER THAN 25 "

Ready     DROP

Ready ELSE

Ready     10 < IF

Ready        PRINT " LESS THAN 10 "

Ready     ELSE

Ready        PRINT " BETWEEN 10 AND 25 "

Ready     THEN

Ready THEN ;
```

One IF - ELSE - THEN is placed between the ELSE and THEN of another one. Note that before the first comparison, we DUPlicat the number because we don't know yet whether or not it will be needed for the second comparison. If the number is greater than 25, then it is not needed again, and is DROPped.

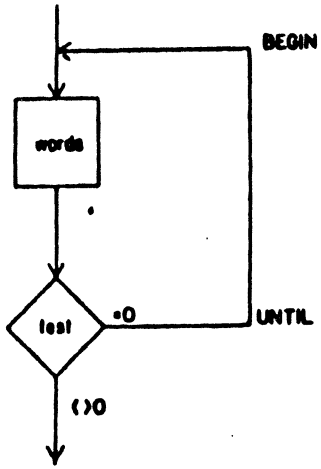
```
Ready -62 TWOLIMITS
LESS THAN 10

Ready 19 TWOLIMITS
BETWEEN 10 AND 25

Ready 684 TWOLIMITS
GREATER THAN 25
```

BEGIN - UNTIL

Another construct that allows repeated execution is BEGIN - UNTIL. The form is:

```
BEGIN
   <words to be repeated>
   <test stack value>
UNTIL
```

The word BEGIN marks the beginning of the construct. The words between BEGIN and UNTIL are executed, then the word UNTIL removes a number from the stack. If the number is zero, then the program branches back and the words between BEGIN and UNTIL are executed again. This loop is repeated until the stack value is nonzero, then the program continues past the UNTIL. This is the flowchart for BEGIN - UNTIL:



The following example starts with a zero on the stack, then prints the number, adds 1 to it, then loops back until the number equals 8:

```
Ready 0 BEGIN DUP . CR 1 + DUP 8 = UNTIL
0
1
2
3
4
5
6
7
[8]
Ready
```

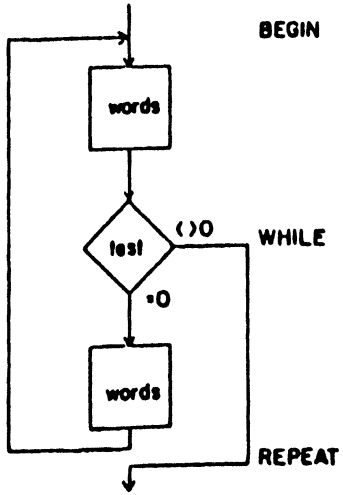
The words "DUP . CR" print the number without losing it and issue a carriage return; "1 +" increments the number; and "DUP 8 =" determines if the number equals 8. Notice that this loop leaves a copy of the number on the stack when it finishes. Adding DROP to the end of the line takes care of this.

BEGIN - WHILE - REPEAT

The BEGIN - WHILE - REPEAT construct is similar to BEGIN - UNTIL. The form is:

```
BEGIN
   <words to be repeated>
   <test stack value>
WHILE
   <controlled words>
REPEAT
```

The word BEGIN again marks the beginning of the construct. The words between BEGIN and WHILE are executed, then WHILE removes a number from the stack. If this number is nonzero, then the controlled words between WHILE and REPEAT are executed, then execution jumps back again to the words after the BEGIN. If the number is zero, then the program jumps directly past the word REPEAT and continues on. The key to remembering this is that the controlled words are REPEATed WHILE the stack value remains nonzero. This is the flowchart for BEGIN - WHILE - REPEAT. Note that the test is at the beginning of the controlled part:



The following example is similar to the previous example for BEGIN - UNTIL. The number is tested first this time. While it is not equal to 8, it is printed and incremented, and the cycle is repeated:

```
Ready 0 BEGIN DUP 8 <> WHILE DUP . CR 1 + REPEAT
0
1
2
3
4
5
6
7
[8]
Ready
```

CASE: - THEN

Sometimes a choice needs to be made from a range of possible numbers. The CASE: construct allows you to do this. The form is:

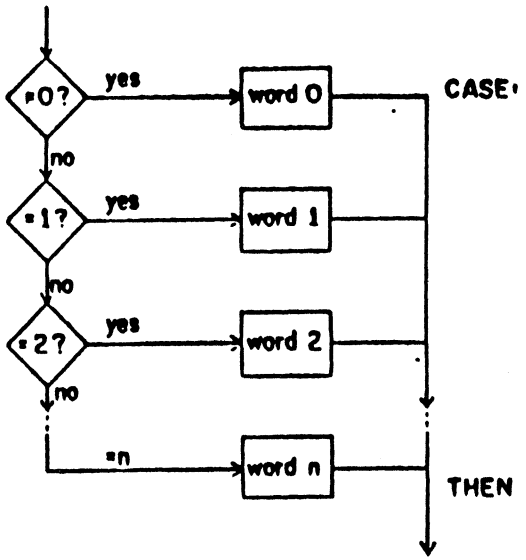
<stack value>

CASE:

<word 0>
<word 1>
<word 2>
.
.
.
<word n>

THEN

The word CASE: removes a number from the stack and uses this number to select and execute a single word from a list of words. A zero selects word 0, a one selects word 1, etc. The word THEN marks the end of the CASE: construct, and is required. The flowchart for CASE: follows on the next page:



The following example shows how CASE: works:

```
Ready : X PRINT " THE NUMBER IS ZERO " ;

Ready : Y PRINT " THE NUMBER IS ONE " ;

Ready : Z PRINT " THE NUMBER IS TWO " ;

Ready : CASE.TEST

Ready CASE:

Ready    X

Ready    Y

Ready    Z

Ready    BELL

Ready THEN ;
```

X, Y, and Z are words we have defined and are called by the word CASE.TEST. The CASE: list in CASE.TEST contains four words, so the construct uses the numbers 0 through 3. Zero selects X, 1 selects Y, 2 selects Z, and 3 selects BELL:

```
Ready 0 CASE.TEST
THE NUMBER IS ZERO

Ready 1 CASE.TEST
THE NUMBER IS ONE

Ready 2 CASE.TEST
THE NUMBER IS TWO

Ready 3 CASE.TEST
(The Apple speaker beeps.)
```

Warning: If the number which CASE: removes from the stack is too large or is less than zero, something strange and probably not-so-wonderful will happen. For example, the system may hang up. (In the above example, the only acceptable numbers for CASE.TEST are 0, 1, 2, and 3.) The key to avoiding trouble is to simply not let numbers out of the CASE: range go into the word CASE:. There are a number of ways to do this. Here is one for the above example:

```
Ready : SAFE.CASE

Ready DUP DUP 3 <= SWAP 0 >= AND

Ready IF

Ready    CASE.TEST

Ready ELSE

Ready    PRINT " THE NUMBER IS NOT BETWEEN 0 AND 4 "

Ready    DROP

Ready THEN ;
```

SAFE.CASE first checks the number to see that it is between 0 and 4 before passing it on to CASE.TEST. If it is out of range, a message is printed. (You may want to try the words "DUP DUP 3 <= SWAP 0 >= AND" directly from the keyboard to see how they work together).

```
Ready 2 SAFE.CASE
THE NUMBER IS TWO

Ready 7 SAFE.CASE
THE NUMBER IS NOT BETWEEN 0 AND 4
```



```
Ready -6 SAFE.CASE
THE NUMBER IS NOT BETWEEN 0 AND 4
```

Program Structure and Other Miscellaneous Thoughts

Notice that in the last example for CASE: above, we began by defining three short words: X, Y, and Z. Then we defined the word CASE.TEST, which calls one of those three words. Finally we defined SAFE.CASE, which calls CASE.TEST.

This "chain" of definitions is the way long programs in GraFORTH are built up. The 'low-level' words, which usually do rather menial tasks, are defined first. Then the next level of words, which call the first set of words, are defined. This process builds layer by layer until one last word is added to the top of the word library, which "coordinates the show". The entire program can be run by simply typing the name of this top word.

The beauty of this scheme is that each level of words can be thoroughly tested and debugged before moving on to the next higher level. This helps to prevent the all-too-familiar scene of the programmer helplessly wading through miles and miles of computer print-out trying to find the elusive "bug" in a program.

Another advantage is that with separate word definitions, you can have more than one "program" in memory at a time. Words can be defined completely independently of each other, and used as individual programs or routines.

....Which brings us back to some specific points on GraFORTH.

Word References

Words in GraFORTH can only be defined in terms of already existing words, which reside in the GraFORTH word library at the time. In fact, any reference to a word that is not currently in the word library will produce an error message, and the unknown word will be ignored:.

```
Ready 5 0 DO I . CR STRANGE LOOP
```

TRAN _ _ot Fo_ _ [Pre_ _ _turn,

0
1
2
3
4
Ready

Another source of trouble is defining a word with the same name as an already existing word. If this happens, the new word is added to the word library, but a warning message is printed:

```
Ready : OVER PRINT " OVER THE RIVER AND THRU THE WOODS " ;

OVER Not Unique (Return)
```

With two words with the same name in the word library, how does the system choose between them? For our example, any words that referenced OVER before the new definition was added will still reference the earlier word. Any new references to OVER will reference the new definition. That means that the original definition is no longer accessible from the keyboard! In general, defining words with existing word names is not a good idea and should be avoided.

Programmers who like to dabble with recursion will be happy to hear that GraFORTH words can call themselves. Word definitions can also be nested one definition inside another, allowing the inside and outside words to call each other. These capabilities are very useful in certain recursive applications, but should be avoided if not needed. (Your programs can get hard for people to follow!)

Speed and Flexibility vs. Error Checking

GraFORTH is a very fast language. It has to be to manipulate 3-D images at the speeds it does. GraFORTH is also very flexible. As you'll see in Chapter 6, GraFORTH gives you direct control of your Apple.

You may be asking, "What's the catch?" The "catch" is that GraFORTH has little built-in error checking. In terms of speed, if your program works correctly, then repetitive error checking schemes can only slow your program down.

In terms of flexibility, if you're allowed to do nearly anything, then there is nothing "to protect you from". GraFORTH follows the Forth convention that if you want error checking, you'll write it into your programs. If you don't need error checking, you don't have to include it.

One example is 'stack underflow and overflow'. Stack underflow is where a word tries to remove a number from the stack and the stack is empty. If this happens, GraFORTH will merely return the number that was last on the stack. Stack overflow is caused by trying to place more than 128 numbers on the stack. If this happens, the extra numbers are ignored. If a stack underflow occurs when the stack display is on, a long stream of stack numbers may be displayed. If this happens, just type ABORT to clear the stack. (The key to avoiding stack problems is to be aware of what is happening on the stack at all times. Sometimes "single-stepping" through a list of words with the stack display on can help.)

Another example of error checking is with words that "expect" a number in a given range. We've seen this already with the word CASE:. Many words in GraFORTH use numbers in a specified range. Some words don't mind the excess; they "fold" the number back into an appropriate range and ignore the difference. Other words (like CASE:) do not fold back, and must be given a valid number. As we introduce words, we'll include any valid ranges.

Words Which Look Forward

Most words in GraFORTH look to the stack for any data or information they might need. Some words, like PRINT or FORGET, look forward down the input line for further data. You might be tempted to build a colon definition like the following:

```
Ready : TESTWORD  CR CR PRINT ;

Ready TESTWORD " HI THERE "
```

Don't try it! The word PRINT looks for the text to be printed as it is compiled, not when it is executed. The above example will not work, and it may cause the system to go off the deep end... The other words (introduced in later chapters) which look to the input line for data work the same way, and should be used as described.

Text vs. Graphics

Since the Apple graphics screen is used for the normal GraFORTH display, mixed text and graphics, changeable character sets, and lower case displays can all be used in GraFORTH. However, text scrolling is not as fast as it would be on a standard text display. GraFORTH includes two words, GR and TEXT, which enable you to switch between the graphics display and a text-only display. The only advantage to using the text display is for faster scrolling, which can occasionally come in handy when editing files from the editor.

Memory Considerations

Because of the large number of features implemented in GraFORTH, and the fact that both graphics screens are being used, free memory for program development is somewhat limited. The presence of a language card or RAM card eases this limitation considerably. The memory map in Appendix B shows the available free memory with and without a language card, and with or without the text editor in memory. Memory considerations when using the text editor will be discussed in the next chapter.

The way to keep memory free is to always FORGET words that are no longer needed. Loading one large program onto the word library above another is a sure way to run out of memory. Be aware of what is on the word library, and how much memory is being used.

There are two words to help you:

The word PRGTOP places the address of the top of the word library on the stack. This can let you know how large things are getting. This example was done with no additional words on the word library. (The addresses printed here are for example purposes only. The address numbers displayed may be slightly different.)

```
Ready PRGTOP .
-32256
```


For people who "think" in hexadecimal, the word $LIST can also be very useful. $LIST is identical to LIST, except that it also displays the hexadecimal addresses of each word in the word library. By comparing adjacent numbers, you can determine how much memory each word takes. Here is a sample of a $LIST:

```
Ready $LIST

$8254 CHS
$8246 ABS
$8224 SGN
$81F4 CALL
$81E9 PREG
$81DE YREG
  .   .
  .   .
  .   .
```

Since $LIST displays the address at which each word begins, the first address shown is the beginning of the top word, not the top of the word library at the end of the word. To determine the address of the top of the word library in hexadecimal, you can define a "dummy" word and then use $LIST. The top address will be the top of the word library after the dummy word is deleted:

```
Ready : IT ;        ("IT" does not execute anything.)

Ready $LIST

$826E IT
$8254 CHS
$8246 ABS
  .   .
  .   .

Ready FORGET IT
```

$826E is the hex address of the top of the word library,

Conclusion

Let's take a break here, and digest some of this information. This might be a good time to grab a pizza, take a nap or come out of hiding and visit someone who hasn't seen you in a few days! Anyway, when you come back we'll move into the text aspects of GraFORTH and introduce you to the supplied GraFORTH text editor. (We'll also show you some wonderful special characters to make your Apple a little more friendly...)

CHAPTER FOUR: TEXT MAGIC

Chapter Table of Contents:

Purpose and Overview

In Chapter 3, we learned (among other things) how to define new words in terms of existing ones. The words were added to the dictionary and could be called at any time. However, there was no way to save the text of the definition; to go back to the string of words which defined it.

Enter the GraFORTH text editor, a straightforward general purpose line-oriented editor. Text can be created here, modified, saved to disk, read back in, and more.

GraFORTH includes words to compile text into the system from the editor or directly from the disk. If any defined words need to be modified, they do not have to be completely re-entered. They can be changed from the editor, then recompiled by the system.

In this chapter, we'll discuss how to use the text editor and how to compile GraFORTH programs from the editor or from disk. We'll also give you some pointers to keep both system and editor memory happy. But first, we should discuss some of the special characters used in GraFORTH, both in and out of the editor, and how they can help both your programming and your programs.

Strange and Wonderful Characters

Upper and Lower Case

If you've looked at the GraFORTH demonstration, you've seen all these lower case characters on your Apple screen, but until now, we haven't told you how to enter lower case characters yourself. There's really no magic, as we'll soon see!

Upper and lower case can be set in a number of ways, and each is a two-key process.

While entering a line, type CONTROL-O, then

"E": Subsequent entries will be in lower case unless ESC is pressed in advance. If ESC is pressed first, the following character will be in upper case.

"S": Entries will be shifted to upper case if your Apple][has the one wire shift key modification. (A wire running from the shift key to the game paddle AN3 input).

"U": All entries will be in upper case.

"L": All entries will be in lower case.

"Hidden Characters"

Although the Apple][keyboard won't accept all the ASCII characters, GraFORTH will. Here are the keys to press to get the "hidden characters":

CONTROL-Shift-N gives a left bracket
CONTROL-Shift-M gives an underline
CONTROL-Shift-P gives a reverse slash
Shift-M gives a right bracket unless one of the lower case shift options has been set.

Cursor Movement

As you may have discovered by now, the Apple arrow keys work as they do in most Apple applications: the left arrow is a "backspace" key that enables you to back up on the line to correct mistakes. The right arrow is a "retype" key. If you use the right arrow key to move the cursor over text on the screen, the text will be treated by GraFORTH as if it were being typed again directly from the keyboard.

The Apple ESCape codes for moving the cursor also work from GraFORTH. These can be handy for making fast corrections from the GraFORTH text editor. If you're unfamiliar with the Apple ESCape codes, we suggest you consult one of the Apple manuals. Most of the manuals discuss these codes.

Note: If any of the lower case shift modes have been set, then the ESCape key cannot be used to move the cursor. To move the cursor using ESCape, first set upper case only (ConTRoL-O, U) shift mode.

Line Insertions

Insertions can be made into the middle of a line using ConTRoL-I. Pressing ConTRoL-I pushes any characters to the right of the cursor one more space to the right.

To make an insertion using ConTRoL-I, first use the Apple ESCape codes to move the cursor to the beginning of the line to be changed. Use the retype key to move the cursor to the point of insertion, then press ConTRoL-I enough times to open up a space in the line for insertion. Now enter the additional text, then use the retype key to move the cursor to the end of the line, and press <return>.

Note: The ConTRoL-I feature works for editing only one 40-character line at a time. Pressing ConTRoL-I too many times can push text off the right end of the screen and into Never-Never Land....

The Text Editor

There are actually two text editors on the GraFORTH system disk, named OBJ.EDITOR1 and OBJ.EDITOR2. The first is used on systems that do not have a language card or RAM card and can edit about 2000 characters without changing the default settings. The second is used with systems that have language cards and can edit about 11,500 characters. Otherwise, the two editors are identical.

Note: GraFORTH and the GraFORTH editor both use standard DOS text files for program storage. If you have a text editor that you are accustomed to that also uses DOS text files, you may use it instead of the GraFORTH editor. Large programs will be more manageable in a text editor such as Apple Writer 2.0. Compiling programs into the GraFORTH system from disk is the same regardless of what editor is used to create the file.

For the editor examples in this chapter, we will use English sentences for text instead of GraFORTH programs. The editor doesn't know the difference, and it makes things easier to read. The editor is of course usually used for writing GraFORTH programs. The GraFORTH word MEMRD, discussed in the next section, allows text to be read and compiled directly from the editor.

To enter the editor from GraFORTH, type EDIT. The appropriate editor will automatically be loaded. In a few seconds you should see the GraFORTH editor header:

GraFORTH][Editor (C) 1981 P. Lutus

The first command to know in the editor is "?", the question mark. Entering a question mark gives you the Editor Command Index, a list of all the other editor commands:

```
?

Save
Get
Insert
Delete
Program
Memory
List
Write
Erase
Autonum
Bye
ConTRoL-D=DOS
```

We'll discuss each of these commands in turn, but first let's find out how to enter text into the text editor.

Line Entries

Entries to the text editor are preceded by line numbers. These line numbers have no meaning to GraFORTH, and are not retained in the program file when it is saved to disk. They simply serve as an index to the file while it is in memory. The editor line numbers are in steps of 10, and whenever insertions or deletions are made, the file is renumbered automatically, in steps of 10 again.

To enter a line, simply type a line number followed by the line. Here are some example lines to enter:

```
10 My very first editor line!
20 Entering lines in the editor is
30 similar to entering lines in Basic.
```

LIST

To see that these text lines have been stored, they can be listed by typing "LIST" or simply the letter "L". (All of the editor commands are single letters, and should be entered in upper case.)

```
L
10 My very first editor line!
20 Entering lines in the editor is
30 similar to entering lines in Basic.

Done
```

(The "Done" message is printed whenever an editor command is successfully accomplished. We're not going to show it in all of our examples, though.)

Inserting lines in the text editor is much the same as in Basic. Simply enter a line number between the line numbers you want the text inserted into. Remember that after the insertion is made, however, the lines will be renumbered in steps of 10. Let's insert a line between line 10 and line 20 by giving it a line number of 15:

```
15 With some important exceptions,
```


Now let's list the file again to see that the line was inserted and the following lines were renumbered:

```
L
10 My very first editor line!
20 With some important exceptions,
30 Entering lines in the editor is
34 similar to entering lines in Basic.
```

If the file being edited gets rather long, you don't have to list the entire file every time. The listing automatically stops every 16 lines. If you press CONTROL-C during the pause, the listing will stop. If you press any other key, the listing will continue.

You can also use "List" to list a single line or a range of lines. Assuming a file contains at least 15 lines (numbered 10 to 150):

```
L 80       lists line 80 only.
L 80,150   lists lines 80 through 150.
L 80,      lists from line 80 to the end of the file.
L ,80      lists from the beginning of the file to line 80.
```

AUTONUM

The editor also provides automatic line numbering. Going back to our original example, list the file, then press "A" for "Autonum". The next line number, line 50, will appear for you. Enter a couple of lines with Autonum on:

```
A
50 This is much nicer than having
60 to enter the line numbers myself.
70
```

To stop the Autonum feature, just press <return> at the beginning of the line after the line number.

To change a line already in the editor file, simply retype the line number followed by the corrected line. The ESCape codes and the right-arrow key can be used to retype a line that is on the screen, and ConTRoL-I can be used to make insertions within the line.

Simply entering a line number followed by <return> won't delete a line, as is true for Basic. Instead this will create a blank line, very useful in its own right for separating program segments and word definitions. To make a blank line while the Autonum feature is in use, enter a space, then press <return>.

DELETE

The "D" ("Delete") command is used for deleting a line or range of lines. Its format is identical to "List" (though its effects are very different!):

```
D 80       deletes only line 80.
D 80,150   deletes lines 80 through 150.
D 80,      deletes from line 80 to the end of the file.
D ,80      deletes from the beginning of the file to line 80.
```

ERASE

To erase the file in memory, press "E" for "Erase". A prompt will appear:

```
Erase (Y/N) :
```

This prompt prevents inadvertent file erasure. Enter "Y" and press Return to erase the file.

Automatic Insertions

In a previous example, we used Autonum to add to the end of the file. When used in the middle of a file, Autonum also automatically inserts the text, making room for the text and renumbering later lines. For these examples, let's start with a new file. Erase the file in memory, then enter a couple of lines:

```
10 The first line in the file...
20 The last line.
```


We can start an insertion by entering the first line number of the insertion ourselves:

```
15 must surely be followed by others.
```

Now, pressing "A" will cause automatic line numbering that starts following the last entered line, line 15, and insert this text into the file. Since line 15 is renumbered to become line 20, the next line number, printed with the Autonum feature, is line 30:

```
A
30 Autonum does more than generate
40 line numbers.  It also inserts
50 into the middle of a file.
60
```

Again, Autonum is turned off by pressing <return> with no text. Let's list the file now:

```
L
10 The first line in the file...
20 must surely be followed by others.
30 Autonum does more than generate
40 line numbers.  It also inserts
50 into the middle of a file.
60 The last line.
```

INSERT

The "I" ("INSERT") command can also be used to initiate insertions into a file. Instead of typing the first inserted line before using Autonum, INSERT is used to specify the starting line number. Let's delete the lines we just entered, and re-enter them, this time using INSERT.

```
D 20,50

Done

L
10 The first line in the file...
20 The last line.
```

We want to insert between lines 10 and 20, so enter:

```
I 15
```

Autonum will use this line number as the point of insertion, instead of the last accessed line.

```
A
20 must surely be followed by others.
30 Autonum does more than generate
40 line numbers.  It also inserts
50 into the middle of a file.
60
```

List the file again, and you will see that these lines have been re-inserted into the file.

SAVE

To save a file to disk, press "S". A prompt will appear:

```
S
(Filename) :
```

Enter the file name you want the file to be saved under. If desired, you can also specify a disk slot and drive number here, separated by commas using the standard DOS format. Here are a couple of examples:

```
(Filename) : TESTFILE
(Filename) : TESTFILE,S6,D1
```

If you want to save only a portion of the file to disk, enter a slash after the filename, followed by the range of line numbers to be saved:

```
(Filename) : TESTFILE/80,150  (Saves lines 80 to 150)
(Filename) : TESTFILE/,80     (Saves beginning to line 80)
(Filename) : TESTFILE/80,     (Saves line 80 to end of the file)
```


GET

To get a file from disk and load it into the editor memory, press "G". A prompt will appear:

```
G
(Filename) :
```

Enter the name of the file to be loaded and, if desired, the disk slot and drive at which it is located, using the same format as SAVE.

To get a file and insert it at a particular location in the existing file, enter a slash after the filename, followed by the destination line number in the current file. This example will insert the file TESTFILE into the current editor file between lines 110 and 120:

```
(Filename) : TESTFILE/115
```

Note: "GET" always inserts the file into the present memory contents. The file contents are not erased by "GET". To erase the present file and get a new one, "ERASE" the present file and then "GET" a new one. Seems simple enough.

Note: Since "GET" and "SAVE" use slashes to specify certain lines in a file, filenames that contain slashes cannot be used with the text editor.

DOS Commands

To enter a DOS command directly from the editor, press CONTROL-D and <return>. A prompt will appear:

```
Enter DOS Command :
```

From this prompt, you can enter any DOS command, to get a catalog, delete files, lock files, etc. The prompt repeats after each DOS command so that you can execute several commands without having to press CONTROL-D every time. To return to the editor prompt (a flashing cursor with no prompt line), simply press <return> twice.

Printing Files

Editor files can be printed directly from the editor. Type ConTROL-D and <return> to get the DOS prompt, then type "PR#1". (If your printer is in another slot, substitute that number.) The printer will be activated, then press <Return> twice to remove the DOS prompt.

With the printer enabled, you can type "L" to list the file to the printer, pressing <return> when the listing stops every 16 lines. A better way is to type "W" for "Write". This option writes the editor file out without any pauses.

Since "PR#0" does not reconnect GraFORTH's special graphic output, press Reset to turn the printer off and return to a normal display. The next chapter includes a discussion on how to access peripherals and return to GraFORTH in a normal manner.

Memory Considerations

As the GraFORTH word library grows, it can begin to use the same memory that is used from the editor. If the word library is large enough, adding words can erase a part of the editor file, or even the editor program itself. Conversely, using the editor can destroy the top of the word library, requiring the system to be rebooted.

In addition, the amount of usable editor file memory is determined by the presence or absence of a language card. We suggest you study the memory map in Appendix D and become generally familiar with areas of memory used by the GraFORTH language, the editor program and the editor file in your system.

To find the amount of free memory left in the editor file area, press "M" for "Memory". You will see:

```
Free Memory
```

followed by the number of bytes (or characters) of memory left.

You may want to adjust the amount of memory used by the text editor, to avoid conflict with GraFORTH. To accomplish this, you may position the file either up or down in memory. To do this, press "P". A display will appear:


```
Program Length
Position
Free Memory
Change Position (Y/N) :
```

The length, position (starting address of the editor file area), and memory labels will be followed by their present numeric values. To change the editor file position, enter "Y" to this option. You will be prompted:

```
Enter New Position :
```

On a language card system, the file position can be moved somewhat higher to make more room for the GraFORTH word library. On a non-language card system, it's often best to use the editor without regard to keeping the word library intact, save the edited file to disk, and reboot GraFORTH from scratch. This method will be outlined in the next section.

Leaving the Text Editor

To leave the text editor and return to GraFORTH, simply type "B" for "Bye".

Program Compilation

GraFORTH normally accepts its input from the keyboard. Each line is compiled immediately and acted upon if necessary.

GraFORTH can also read lines from the editor file or from a disk file, treating the lines as if they were typed from the keyboard. GraFORTH programs can be written in the editor and saved to disk, then read and compiled into the system.

The word to read and compile text from the editor buffer is MEMRD. MEMRD removes a number from the stack, interprets this number as an address, and begins reading text from memory starting at this address. It reads and compiles until it either reads a zero byte (marking end-of-file) or encounters an error. Control is then returned back to the keyboard.

The address of the editor file buffer is 34817, unless changed with the Program Position option in the editor. To read the text from the editor, type:

```
Ready 34817 MEMRD
```

To read and compile directly from a text file, the word READ is used. The form for READ is:

```
READ " <filename> "
```

READ reads to the end of a file, or until an error is encountered.

MEMRD and READ are usually used to compile word definitions into the word library, but immediate-execution lines can also be included.

Comments

Usually, the GraFORTH Editor is used for writing and editing GraFORTH programs instead of the text used earlier in this chapter. Comments in the source file of a GraFORTH program can often be very helpful for understanding and keeping track of long programs.

The GraFORTH word "(" is available for inserting comments into program files. In compiling the program, when GraFORTH sees a "(" set off with a space on either side, it ignores the rest of the text on the line until it sees a ")". Comments can be inserted freely in the source file. Here is an example of such a comment line:

```
10 ( PARENTHESES AROUND A COMMENT )
```

Using the Editor with GraFORTH

When smaller programs are being developed, the editor and the GraFORTH system can be used closely together. Load the editor and enter the program, then return to GraFORTH and compile the program with MEMRD. If the program has bugs or needs further changes, simply return to the editor and make those changes. When returning to GraFORTH, FORGET the original word definitions before compiling the new ones, to prevent "Not Unique" errors from occurring. (Unless you're testing a very short program, you should also save the program to disk after each edit.)

When larger programs are being developed and GraFORTH/editor memory conflicts are likely, it's best to separate editing and compiling. Use the editor to write the program, then save the program to disk. Then return to GraFORTH and compile the program with READ or MEMRD. If the program needs to be changed, FORGET the words before returning to the editor, so that editor usage won't erase the top of the word library. From the editor, reload the program from disk and continue editing.

Understanding and following the above guidelines will protect you from memory conflicts, and will make programming in GraFORTH much easier.

As you become more comfortable with programming in GraFORTH, you will probably want to use the editor to list some of the program files on the system diskette. We encourage you to do this. The system files provide excellent programming examples in GraFORTH.

CHAPTER FIVE: DELVING DEEPER. . .

Chapter Table of Contents:

Purpose and Overview

Chapter 4 introduced GraFORTH as a language. In this chapter, we'll round out the language and give you some of the background you need before moving on to the graphics features ("What? You mean this language has graphics too?!") in the next three chapters.

We'll start off by introducing the GraFORTH standard text manipulation words (not to be confused with the fancy ones we'll show you in Chapter 7). Then we'll discuss storing data in memory, and the various words used to accomplish this. We'll talk about the two other kinds of words in GraFORTH (variables, and strings), and how they can be used to set aside memory for data storage in very convenient ways. Following this will be a discussion of the string operators built into the system and on a disk file.

Next, we'll talk about using DOS from GraFORTH, and introduce SAVEPRG, the word that makes your work permanent. We'll tie up the loose ends with a number of words which are extremely useful, but evade strict categorization.

Text Formatting Words

These are the words which are used to position text and characters on the screen, and clear the screen, or portions of it. Each of these words is straightforward.

Review

You have seen how to use PRINT, SPCE, and CR already in Chapter 3. For a quick review...

PRINT prints following quoted text starting at the current cursor position.

CR issues a carriage return, moving the cursor to the beginning of the next line.

SPCE prints a space.

New Text Positioning Words:

HTAB removes a number from the stack, interprets it as a horizontal cursor position, and tabs to that cursor position. The cursor remains in the same vertical position.

VTAB removes a number from the stack, interprets it as a vertical cursor position, and tabs to that cursor position. The cursor remains in the same horizontal position.

The valid ranges for HTAB and VTAB depend on the current character size (CHRSIZE), which will be discussed in Chapter 7. For the normal character size we are using now, the range for HTAB is 0 to 39, and the range for VTAB is 0 to 23.

WINDOW removes four numbers from the stack to establish a text window. The text window is a rectangular area on the screen designed to protect other parts of the screen from being overwritten. All text scrolling will occur inside the window, leaving the rest of the screen unaffected. The form for WINDOW is:

<left> <width> <top> <bottom> WINDOW

Left, top and bottom are actual margins for the window. Width specifies the right margin as the number of characters from the left margin. The bottom margin number should reference the line immediately below the window. For example, a window 10 characters wide by 5 lines high in the lower right corner of the screen would be set by:

```
Ready 30 10 19 24 WINDOW
```

(The left margin is at position 30, the window width is 10 characters, the top margin is at line 19, and the bottom margin is above line 24.)

HOME erases the screen and homes the text window.

CLEOP (CLear to End Of Page) erases the screen from the current cursor position to the end of the text window.

CLEOL (CLear to End Of Line) erases from the current cursor position to the end of the line.

ERASE erases the entire screen, regardless of the setting of the text window. ERASE is usually faster than HOME.

Data Storage and Retrieval

GraFORTH has the capability of examining and changing the value stored in any location in memory. If desired, the actual decimal memory address can be entered from the keyboard for storage or retrieval. We'll show you data access in this way first, and then discuss an easier technique using named variables.

GraFORTH Memory Addresses

The Apple][contains 65536 addressable "locations". These locations are usually numbered from 0 to 65535. Most of them are used for RAM memory, which can be either read from or written to. Each memory location can store one 8-bit 'byte', representing a number from 0 to 255. Two locations, or two bytes, can store a number from 0 to 65535. Since two bytes can only reference positive numbers in the range 0 to 65535 and people sometimes like to use negative numbers, one 'bit' of the number is used to tell us the numbers sign, positive or negative. Therefore, GraFORTH uses a number range of -32768 to 32767. Since it is desirable that zero in both systems be zero, a "wrap-around" scheme is used: Addresses above 32767 are treated as negative numbers, and continue to increase until they again reach zero. (This is identical to the system used by Apple's Integer Basic, where a call to enter the system monitor must be done with a negative number: CALL -151.) A diagram will best explain this:

Positive Decimal Addresses	GraFORTH Decimal Addresses
0	0
1	1
2	2
.	.
.	.
32766	32766
32767	32767
32768	-32768
32769	-32767
32770	-32766
.	.
.	.
65533	-3
65534	-2
65535	-1

Notice that both address ranges continually increase, except that the GraFORTH addresses have a transition in the middle from positive to negative numbers. The memory map in Appendix D includes GraFORTH decimal addresses and hexadecimal addresses.

Storage and Retrieval Words

To store a number directly into a desired memory location, simply place the number you want to store and the address where you want it stored on the stack. Then type "POKEW". The word "POKEW" stands for "poke-word" and removes two numbers from the stack, interpreting them as value and address, and stores the data value at the given location. Since GraFORTH numbers occupy two bytes (commonly called a 'word', not to be confused with GraFORTH words), it actually uses the given location and the one immediately after it.

This example stores the number 12345 at location 2816 (which happens to be the beginning of a large free area of memory in GraFORTH):

```
Ready 12345 2816

[12345]
[2816]
Ready POKEW

Ready
```

To recall a number from memory and place it on the stack, place the address of the desired memory location on the stack and type "PEEKW". The word "PEEKW" stands for "peek-word" and removes a number from the stack, interprets it as an address, retrieves a number from that address, and places the retrieved number on the stack. The following example recalls the number we just stored in memory:

```
Ready 2816

[2816]
Ready PEEKW

[12345]
Ready
```

To store a single-byte value to one memory location, the word "POKE" is used instead of "POKEW". The form is the same. This example stores the number 255 to location -28721:

```
Ready 255 -28721 POKE
```

The word "PEEK" is used to retrieve single bytes from memory. The form for "PEEK" is the same as for "PEEKW". This example reads a special Apple location that contains the current horizontal cursor position:

```
Ready PRINT " Demonstrating PEEK "  36 PEEK
Demonstrating PEEK

[18]
Ready
```

Printing the phrase "Demonstrating PEEK" moved the cursor out to position 18. Reading location 36 retrieved this position as a number.

To summarize, here is a table of the four storage and retrieval words:

Word	Before	After	Description
POKEW	m n -	- - -	Puts two byte m into location n
PEEKW	n -	m - -	Reads two byte m from location n
POKE	m n -	- - -	Puts one byte m into location n
PEEK	n -	m - -	Reads one byte m from location n

Variables

GraFORTH allows you to set aside space for number storage through the word "VARIABLE". VARIABLE creates a new word and places it on the GraFORTH word library. VARIABLE has two forms; the first one is:

VARIABLE <variable name>

The variable name is the name of the word created and placed on the word library. For example:

```
Ready LIST

CHS
ABS
SGN
 .
 .
 .

Ready VARIABLE TEMP

Ready LIST

TEMP
CHS
ABS
SGN
 .
 .
```

The new ... TEMP ...nsis ... two ...s: a ...-byte ...ce set aside for storing a number, and a call to an internal GraFORTH routine that either places the value of the variable on the stack or stores the stack value into the variable.

To store the number 12345 in TEMP, type:

```
Ready 12345
[12345]

Ready -> TEMP

Ready
```

The GraFORTH word "->" is a special word that says "store into". It is created by typing a minus sign '-' followed by a right arrow '>'. This word sets an internal flag used by variables to determine if a "store" or a "recall" operation is to take place. When the "->" word is executed it sets this flag so the next referenced variable will do a store, rather than a recall. Note that the variable will clear this flag so no special operator is needed when doing a recall.

Therefore, to recall the value just stored in the variable TEMP, just type its name:

```
Ready TEMP
[12345]
```

Whenever you need to recall the value of a variable, simply type its name. To store a value into a variable, always type the GraFORTH word "->" before typing the variable name.

Unless otherwise specified, when a variable is first created and compiled using the word VARIABLE, the initial value of the variable is zero. To give a variable a different initial value, the other form of VARIABLE is used, where the initial value is entered on the line with the declaration:

<initial value> VARIABLE <variable name>


```
Ready 35 VARIABLE COUNT
```

COUNT will contain the value 35 until another value is stored over it:

```
Ready COUNT .
35

Ready 87 -> COUNT

Ready COUNT .
87
```

We should bring up something important here. The word VARIABLE (as well as STRING, which we'll discuss shortly) is a compiling word, in that it produces new words itself. It is also a word that looks forward down the input line for the word name. It therefore must be used with more care than most GraFORTH words.

To be specific, a VARIABLE declaration cannot appear inside of a colon definition. It must be alone on its own line, not mixed with other GraFORTH words. Any initial value provided when the variable is declared is taken directly from the input line, not from the stack. Since the initial value is not from the stack, it can't be a computed number. For example, the following line will not work:

```
Ready 25 7 * VARIABLE THING
```

Strings

Strings in GraFORTH are words with space set aside for storing characters or text, rather than numbers. Strings are used whenever input is requested from the keyboard, or text has to be manipulated in any way. String words are created with the word STRING, and a number of words devoted to manipulating strings and character data are included in GraFORTH. Additional words, for more complex string tasks, can be found on a disk file called "STRING WORDS", and can be compiled into the word library at any time.

Defining Strings

The word "STRING" is used to create words in the GraFORTH word library that are used for string storage. The form for the word STRING is:

<string size> STRING <string name>

The string name is the name of the word to be added to the word library. The string size is a number specifying the number of bytes, or characters, the string will hold. Remembering how precious computer memory is, the string size should be just large enough to hold whatever string data is expected to go into the string. On the other hand, sufficient room must be allotted in the string for any value ever stored into it. If you attempt to store too much text into a string, you will actually damage the GraFORTH word library. This will force you to reboot the entire system from scratch! To increase speed, FORTH implementations (GraFORTH included) typically do very little error checking. Therefore it is up to you to determine beforehand the size string you will need.

Similar to variables, string declarations draw both their string name and string size from the input line, and have the same restrictions for use as variable declarations.

The following example creates a new word called TESTSTRING which can store a string up to 45 characters long:

```
Ready 45 STRING TESTSTRING

Ready LIST

TESTSTRING
CHS
ABS
SGN
 .
 .
```


GraFORTH strings are indexed from 0 to the string size-1. When a string word is executed, the word removes a number from the stack, adds this number to the address of the beginning of the string, then places this address on the stack. Note that strings differ from variables in that a variable actually places its value on the stack, while a string places the address of the beginning of the string plus the specified index on the stack. Getting the address instead of the value of the string may not seem like much fun, but in a moment we'll show you some powerful words to move string information around!

In the following example, entering "0 TESTSTRING" will place the address of the beginning of the string on the stack. Entering "5 TESTSTRING" will place the address of character number 5 in TESTSTRING on the stack. The last character position of TESTSTRING is accessed with "44 TESTSTRING". Any portion of the string can be accessed quickly in this way.

```
Ready 0 TESTSTRING .
-32241

Ready 5 TESTSTRING .
-32236

Ready 44 TESTSTRING .
-32197
```

Notice the addresses returned are negative. If you don't understand why, be sure to turn back a few pages to the discussion of GraFORTH memory addresses!

Note: The addresses we show are for example purposes. The actual values may be slightly different.

Using Strings

In this section, we'll show you how to use those memory addresses that strings leave on the stack. We'll ASSIGN text to a string, and WRITE and READ lines of text to and from the Apple's screen and keyboard.

To store text directly into a string (or anywhere in memory), the word "ASSIGN" is used, with the form:

<string address> ASSIGN " <quoted text> "

ASSIGN removes a number from the stack, interprets it as a memory address, then stores the text between the quotes into memory starting at that address. Usually the address is supplied by entering the name of a string before typing ASSIGN. Here is an example:

```
Ready 0 TESTSTRING
[-32241]

Ready ASSIGN " SHE SELLS SEASHELLS "

Ready
```

The phrase "SHE SELLS SEASHELLS" has been stored into the string TESTSTRING.

To write the contents of a string to the screen, the word "WRITELN" is used. WRITELN removes a number from the stack, interprets it as a memory address, then writes the text starting at that address to the screen. The form of WRITELN is:

<string address> WRITELN

The following example writes the contents of the string TESTSTRING to the Apple screen:

```
Ready 0 TESTSTRING WRITELN
SHE SELLS SEASHELLS
```

Text can be read in from the keyboard and stored in a string (or anywhere in memory) using the word "READLN". READLN removes a number from the stack, interprets it as a memory address, then reads a line of text from the keyboard and stores the text in memory starting at that address. Like WRITELN, the form of READLN is:

<string address> READLN

Here is an example:

```
Ready 0 TESTSTRING READLN
SEASHELLS                    (You type this line)
Ready
```

The phrase "SEASHELLS" has been read into the string TESTSTRING.

```
Ready 0 TESTSTRING WRITELN
SEASHELLS
```

Of course, assigning, reading and writing don't have to start at the beginning of a string. Strings can be modified by reading into the string, but starting in the middle of the string:

```
Ready 3 TESTSTRING READLN
SHORE

Ready 0 TESTSTRING WRITELN
SEASHORE
```

The word "SHORE" was read into TESTSTRING, starting at character number 3, over the top of "SHELLS".

```
Ready 2 TESTSTRING WRITELN
ASHORE
```

The string was printed starting with character number 2, leaving only the "A" in "SEA".

When a string is stored in memory using ASSIGN or READLN, a carriage return is placed after the last character, marking the end of the string. When WRITELN writes a string from memory, it starts at the specified address and continues until it finds either a carriage return or a byte containing a zero. Either of these mark the end of a string for WRITELN.

String Conversion

Sometimes a string will contain a number stored as text. The GraFORTH word "GETNUM" is used to read the number from the text, placing the value on the stack. GETNUM removes a number from the stack, again interpreting it as a memory address. It then reads the text starting at that address and attempts to find a number, which it places on the stack.

In the following example, the number 321 is first read into a string as text, then converted to a stack value with GETNUM:

```
Ready 0 TESTSTRING READLN
321

Ready 0 TESTSTRING GETNUM
[321]
```

When using GETNUM, nonnumeric characters may follow the number without interfering with the conversion, but the number must begin as the first character of the string.

If GETNUM cannot find a number at the given string address, it places a zero on the stack. To determine for certain whether or not the string-to-number conversion was successful, the word "VALID" is used. VALID leaves a number on the stack. If the last GETNUM was successful, the number will be nonzero; if the conversion failed, VALID will return zero:

```
Ready 0 TESTSTRING READLN
555
Ready 0 TESTSTRING GETNUM .
555
Ready VALID .
253
```

(VALID is nonzero since GETNUM was able to convert the number.)

```
Ready 0 TESTSTRING READLN
YOU CALL THIS A NUMBER??
Ready 0 TESTSTRING GETNUM .
0
Ready VALID .
0
```

(VALID is zero since GETNUM failed to find a number.)

PAD: The System String

GraFORTH includes a predeclared temporary string space of 124 characters called PAD. PAD is convenient for reading keyboard input without having to define a string first.

Actually, PAD is two things: a 124-byte free area of memory used for storing string data, and a word in the GraFORTH word library named PAD which places the address of this free area of memory on the stack. Note that the usual string indexing is not used with PAD:

```
Ready PAD
[812]
```

(812 is the address of the PAD string buffer.)

```
[812]
Ready READLN
Goin' back to my pad.

Ready PAD WRITELN
Goin' back to my pad.
```

To access the middle of the PAD buffer, simply add an offset to the address:

```
Ready PAD
[812]
Ready 6 +
[818]
Ready WRITELN
back to my pad.
```

Note: PAD is considered a temporary string space because the same space is used by the GraFORTH system when compiling words onto the word library, overwriting the previous contents of PAD. Predeclared strings should be used for more permanent string storage.

Accessing Individual Characters in Strings

Since each character in a string occupies one memory location, individual characters in strings can be accessed using PEEK and POKE. In this example, a line of text is placed in TESTSTRING, then the ASCII value of the first character is read onto the stack:

```
Ready 0 TESTSTRING ASSIGN " String pickings "

Ready 0 TESTSTRING PEEK
[211]
```

211 is the ASCII value for the letter "S". "0 TESTSTRING" placed the address of the first character of the string on the stack, then PEEK read the value from this address. In the next example, the "i" in "string" is overwritten with the letter "0" by storing its ASCII value:

```
Ready 239  3 TESTSTRING POKE

Ready 0 TESTSTRING WRITELN
Strong pickings
```


String Words on Disk

There is a file on the GraFORTH system diskette called "STRING WORDS". This file contains additional words for manipulating strings in more complicated ways. To make the string words active, simply compile the file into memory by typing:

```
Ready READ " STRING WORDS "
```

Here are the words in the file "STRING WORDS":

END? is called by a few of the other words to determine if the end of a string has been reached. It removes an address from the stack, reads the value from that address, and returns a 1 if the value is 0 or 141 (the ASCII value for a carriage return), or returns 0 otherwise.

LENGTH removes a string address from the stack and returns the length (number of characters) of that string:

```
Ready PAD ASSIGN " How long am I? "

Ready PAD LENGTH
[14]
```

Remember that string indexing starts at 0 and ends at the string length-1, so the last character of the above string is character number 13.

LEFT$ is similar to the Applesoft "LEFT$" function. The form for LEFT$ is:

<source> <destination> <# of characters> LEFT$

LEFT$ copies the given number of characters from the source string to the destination string. In the following example, the string TESTSTRING is read, then the first 5 characters of TESTSTRING are assigned to PAD:

```
Ready 0 TESTSTRING READLN
ELIZABETH

Ready 0 TESTSTRING  PAD  5  LEFT$

Ready PAD WRITELN
ELIZA
```

RIGHT$ is similar to Applesoft's "RIGHT$". The form is the same as for LEFT$, however the given number of characters are copied from the right end of the string. Continuing from the previous example, 4 characters from the right end of TESTSTRING are now assigned to PAD, overwriting its previous contents:

```
Ready 0 TESTSTRING  PAD  4  RIGHT$

Ready PAD WRITELN
BETH
```

Notice that with GraFORTH's string indexing, the Applesoft function "MID$" can be duplicated with LEFT$. This example reads 3 characters from TESTSTRING starting with the character number 1 (not 0):

```
Ready 1 TESTSTRING  PAD  3  LEFT$

Ready PAD WRITELN
LIZ
```

MOVELN simply copies a string from one location to another. The form is:

```
<source>  <destination>  MOVELN
```

The following example copies the contents of TESTSTRING to PAD:

```
Ready 0 TESTSTRING  PAD  MOVELN

Ready PAD WRITELN
ELIZABETH
```

CONCAT concatenates two strings together. The form for CONCAT is:

```
<string1>  <string2>  CONCAT
```

CONCAT copies the contents of string2 to the end of string1. The contents of string2 are unchanged. In this example, strings are read into both PAD and TESTSTRING, then CONCAT is used to combine the strings in PAD:


```
Ready PAD READLN
GraFORTH:

Ready 0 TESTSTRING READLN
The Apple Graphics Language

Ready PAD  0 TESTSTRING  CONCAT

Ready PAD WRITELN
GraFORTH: The Apple Graphics Language
```

COMPARE makes an alphabetical comparison between two strings, returning a value on the stack. The form for COMPARE is:

```
<string1>  <string2>  COMPARE
```

If string1 is greater than string2 (in alphabetical order, string1 comes after string2), COMPARE returns a 1. If string1 is less than string2, COMPARE returns a -1. If the two strings are equal, COMPARE returns a 0. Here is an example:

```
Ready PAD ASSIGN " LIST "

Ready 0 TESTSTRING ASSIGN " LOST "

Ready PAD  0 TESTSTRING  COMPARE
[-1]
```

The word COMPARE returned a -1 on the stack because the contents of PAD is "less than" the contents of TESTSTRING.

Words Manipulating Individual Characters

GraFORTH also contains words that print individual characters to the screen, and get individual characters from the keyboard. These words interpret numbers as the ASCII values for characters. (A table of ASCII characters can be found in Appendix D.)

The GraFORTH word "PUTC" (PUT Character) prints a single character to the screen. PUTC removes a number from the stack, interprets it as the ASCII number for a character, and prints the character at the current cursor position:

```
Ready 193       (193 is the ASCII value for the letter "A".)
[193]
Ready PUTC
A
```

PUTC removed the 193 from the stack and printed the character "A".

The GraFORTH word GETC (GET Character) places a flashing cursor on the screen, waits for a character from the keyboard to be entered, then places its ASCII value on the stack:

```
Ready GETC

(Type the character "B".)

[194]

(GETC returns 194, the ASCII value for the character "B".)
```

To print a character read in with GETC, simply DUPlicate the value read, and write it to the screen with PUTC:

```
Ready GETC DUP PUTC

(Type the character "Y".)

Y
[217]

(217 is the ASCII value for the character "Y".)
```

To check if a key has been pressed without stopping to wait, "GETKEY" and "CLRKEY" are used. GETKEY and CLRKEY directly use the Apple's special keyboard memory location.

When a key is pressed, its Apple ASCII value is stored in the Apple keyboard location. If a key has been pressed, the number in this location is always 128 or greater. GETKEY reads this location and places its value on the stack. Executing CLRKEY forces the value in the keyboard location to less than 128. The next keypress after CLRKEY is executed will again bring the value to 128 or greater.

Thus, to read the keyboard using GETKEY and CLRKEY, first execute CLRKEY to make the keyboard location less than 128, then use GETKEY until the returned value is 128 or greater. This number will be the ASCII value for the key that is pressed. GETKEY can be interspersed with other tasks so that other things can occur while simultaneously reading the keyboard. Here is a simple example that uses GETKEY and CLRKEY to "grab a character":

```
: GRAB.CHAR
  CLRKEY
  BEGIN
    GETKEY DUP
    128 <
  WHILE
    DROP
  REPEAT
  CLRKEY ;
```

Using Numbers in Other Bases

GraFORTH can accept and display number in bases other than base ten. Four words (HEX, BINARY, DECIMAL,and BASE) allow you to select what base GraFORTH uses.

The word "HEX" causes GraFORTH to read and print numbers in hexadecimal, base 16. In this example, a number is placed on the stack, then base 16 is selected using HEX.

```
Ready 45
[45]

Ready HEX
[2D]
```

Similarly, the word "BINARY" selects base two:

```
Ready BINARY
[101101]
```

The GraFORTH word DECIMAL gets us back to familiar territory:

```
Ready DECIMAL .
45
```

The word "BASE" can be used to select any base. BASE acts as a variable: the word "->" is used to assign the base. The following selects base 8 (octal):

```
Ready 8 -> BASE
```

Note that since BASE is a variable, its current value can be read and displayed. However, any base value displayed in its own base is "10". For example, a 2 in base 2 is 10, and a 16 in hexadecimal is also 10. Thus, to print the base, you must place its value on the stack, change BASE to some other base, then print the stack value. In this short example, the base selected above is displayed before and after changing back to decimal:

```
Ready BASE
[10]

Ready DECIMAL
[8]
```

Because hexadecimal and some other base numbers use letters of the alphabet as digits, possible conflicts between numbers and word names may occur. For example, in hexadecimal, is "ACE" a GraFORTH word name or a number? To help prevent this, GraFORTH allows dollar signs ("$") to precede numbers:

```
Ready HEX

Ready $ACE
[ACE]
```

Note: All of the examples in this manual have assumed that base ten is selected. In addition, some of the programs on the GraFORTH system disk have number formatting that requires base ten. You are free to use other bases, but the results may be quite unpredictable!

Using DOS From GraFORTH

DOS Commands

Using the Apple Disk Operating System from GraFORTH is much the same as from Basic. DOS commands can be called directly from GraFORTH, either from the keyboard or in a word definition. DOS responds to a command that has been preceded by a carriage return and a CONTROL-D (ASCII number 132). (See the Apple DOS manual for more information on disk access in general.) The form for a DOS command from GraFORTH is:

```
CR 132 PUTC PRINT " <DOS command> " CR
```

"CR" prints a carriage return and "132 PUTC" prints a CONTROL-D. The DOS command is printed next, and the line is ended with another carriage return. Here is an example that prints a catalog:

```
Ready CR 132 PUTC PRINT " CATALOG " CR
```

Using Data Files

Text file access is also similar to Basic. The file is opened using standard DOS commands, and data can be read from or written to the file using READLN or WRITELN. File access can be simplified by defining file words ahead of time. For example, to begin reading from a text file, you can use a word like OPEN.READ. (The filename has been stored in PAD.):

```
: OPEN.READ
  CR 132 PUTC PRINT " OPEN " PAD WRITELN CR
  CR 132 PUTC PRINT " READ " PAD WRITELN CR ;
```

After executing this word, the file will be opened for reading, and data can be read in using READLN. At the end of the text, the file can be closed by simply using the GraFORTH word "CLOSE". CLOSE closes any open file.

Since GraFORTH does not have a function similar to Applesoft's "ON ERROR GOTO", DOS errors, including End Of Data, will produce an error message and stop the program. This means that either the length of the file must be known ahead of time, or there must be a special marker at the end of the file so that no more data will be read by the program. The last character in the file must also be a carriage return.

Here is a sample file that makes use of a special End Of File marker. The marker used here is an asterisk on the last line:

```
This is my test file.
Each of these lines will be printed
by the program below.
The last line must be a special marker
to end the file. Here it is:
*
```

Let us say that we have saved this file with the name "TEST". Here is a program that will read and print each line in the file, and will stop when it encounters the end marker "*":


```
: READER
  PAD ASSIGN " TEST "      (Place filename in PAD and call)
  OPEN.READ                (OPEN.READ from above to open file.)
  BEGIN
     PAD READLN            (Read a line from file.)
     PAD PEEK              (Get first character from line.)
     170 <>
  WHILE                    (WHILE this character is not "*": )
     PAD WRITELN           (Write the line to the screen, and)
  REPEAT                   (REPEAT back for the next line.)
  CLOSE ;                  (Close the file.)
```

As the special GraFORTH DOS allots only one file buffer, only one file can be open at a time. The DOS commands "PR#n" and "IN#n" (where n is a number from 1 to 7) can be used from GraFORTH to route data to and from peripheral cards in the back of the Apple. In this way, program text or data can be sent to a printer or other peripheral. After using "PR#n" or "IN#n", either the GraFORTH word GR or TEXT can be typed to re-establish the standard GraFORTH I/O. Do not attempt to use "PR#0" as it will not leave GraFORTH intact.

The following word will print the text in the editor buffer to a printer in slot 1. It reads the characters one at a time and prints them out until it finds a zero byte, marking the end of the editor file.

```
: PRINT.BUFFER
  CR 132 PUTC PRINT " PR#1 " CR
  34A17
  BEGIN
     DUP PEEK DUP
     0 <>
  WHILE
     PUTC
     1 +
  REPEAT
  GR ;
```

Program Control Words

RUN

The GraFORTH word RUN automatically executes the top word on the dictionary. This can be a great convenience when loading and running programs from disk. By using RUN, you don't have to check what the top word on the dictionary is after compiling a file in order to run it. In addition, if the top word has a name something like:

SUPER.ZAPPO.ELECTRO.BLASTERS.APPLE.VIDEO.GAME,

using RUN can save a bit of typing, too....

AUTORUN

The word AUTORUN goes a step beyond this. AUTORUN removes a number from the stack. If this number is nonzero, then GraFORTH will automatically execute the top word on the dictionary every time program control is returned to the GraFORTH system level (i.e. whenever you expect to see a "Ready" prompt). DOS errors, GraFORTH or machine language errors, executing the word ABORT, or pressing the Reset key with the AUTORUN option on will all cause the top dictionary word to be executed. Here is an example to give you a feel for the way AUTORUN works:

```
Ready : TEST PRINT " AUTORUN IS ON!!! " ;
```

We've added this word to the top of the dictionary so that AUTORUN will have a very visible effect.

```
Ready 1 AUTORUN
AUTORUN IS ON!!!

Ready 3 5
AUTORUN IS ON!!!
[3]
[5]
Ready SWAP
AUTORUN IS ON!!!
[5]
[3]
```



```
Ready ABORT

(The screen clears.)

GraFORTH ][ (C) 1981 P. Lutus
AUTORUN IS ON!!!
Ready
```

Fortunately, the AUTORUN option can be turned off by typing:

```
Ready 0 AUTORUN

Ready
```

If the top dictionary word runs a "closed" program which never exits to the system level, the AUTORUN option effectively makes the GraFORTH language itself inaccessible. Any errors or ABORTs simply restart the program.

Saving the GraFORTH System

The GraFORTH language is stored on the system disk as an executable binary file with the name "OBJ.FORTH". As mentioned in Chapter 3, when the disk is booted, this file is automatically loaded and run.

The GraFORTH word SAVEPRG is used to create GraFORTH binary files similar to OBJ.FORTH. SAVEPRG saves the current GraFORTH system, including any new words added to the dictionary, as a binary file. Once created, this file can be BRUN at any time, bringing the modified GraFORTH system back into memory.

SAVEPRG is a powerful tool. You can save "customized" systems, with your favorite special-purpose words already in the dictionary when the system is booted. You can also save finished applications programs, in such a way that the program automatically starts up when booted. This is ideal for games applications, where the obvious presence of a "language" is neither needed nor desirable.

To use SAVEPRG, first compile the words to produce the "finished" system you want to save, then type SAVEPRG:

Ready SAVEPRG

SAVE FILE NAME :

This prompt asks for the filename you want the new system saved as. The GraFORTH system disk automatically RUNs the file "OBJ.FORTH", so if you want this new system to boot automatically, you should name your file "OBJ.FORTH" too. Your file will then overwrite the supplied GraFORTH system. (Make sure you're using a copy of the disk and not the original!) You are then prompted:

AUTORUN (Y/N) :

This prompt asks whether or not you want the saved system to boot up with the AUTORUN option on. If you answer Yes to this question, then the new system will automatically run the top word on the dictionary, starting a program in motion. If desired, your program can later turn the AUTORUN option back off, returning access of the GraFORTH language to the user. If you answer the AUTORUN question with No, the new system will display the "Ready" prompt on boot-up, with immediate access to the language.

After answering this question, this disk whirs for a bit, saving the new system to disk.

Note: As discussed in Chapter 2, a slightly modified version of DOS is used with GraFORTH. Any system saved with SAVEPRG requires this version of DOS to be in memory. New systems should be saved to a copy of the GraFORTH disk, so that the special DOS will be present.

The GraFORTH system as supplied includes an additional word on the top of the dictionary which asks the demonstration prompt on boot-up. This word can be found in the disk file "QUERY". The system was saved with the AUTORUN option on, so that the demo prompt would come up automatically. When you answer No to the demo question, the word turns AUTORUN off (freeing the system), then FORGETs itself! This leaves the system in its "usual" state.

The GraFORTH system can be saved to disk without the demo prompt simply by using SAVEPRG with no additional words on the word library. (This should only be done to a copy of your disk, in case lightning decides to strike while the system is being written to disk.) Boot the disk, answer No to the demo question, then type:

Ready SAVEPRG

SAVE FILE NAME :OBJ.FORTH

AUTORUN (Y/N) :N

After the disk stops whirring, turn your Apple off, then on again. When the system boots, the demo prompt will be gone.

You can also put the demo prompt back into the system. Type:

Ready READ " QUERY "

This adds the word that asks the demo question to the top of the dictionary. Now type:

Ready SAVEPRG

SAVE FILE NAME :OBJ.FORTH

AUTORUN (Y/N) :Y

The system will be saved with the demo prompt back in.

Overlays

GraFORTH programs can automatically load and run other GraFORTH programs, and even delete themselves to free up memory. Program segments that overwrite each other in this way are often called "overlays". The GraFORTH demonstration programs use overlays extensively.

To execute an overlay, include a word in the first file that reads the overlay. Make the first line in the overlay FORGET the words already in memory, and the last line in the overlay file the word RUN. To be more specific:

When you need an overlay, execute a READ <filename>, where <filename> is the name of the overlay. This file will now be read into memory, but since the first line of the overlay contains a FORGET <wordname>, where wordname is the name of the GraFORTH word you wish to forget back to (inclusive), the original file (or portion thereof) will be removed. As reading of the overlay continues, it will now fill memory previously occupied by the original file.

We urge you to examine the demonstration file listings as an example of overlays. Since the FORGET at the beginning of each file does not cause an error if the word being forgotten does not exist, the demo files (or any overlay) can also be directly loaded and run.

Moving Memory

MOVMEM simply moves a block of memory from one location to another. MOVMEM removes three numbers from the stack. The form for MOVMEM is:

<source> <destination> <# of bytes> MOVMEM

The <source> number is the starting address of the data to be moved. The <destination> is the address of where the block is to be moved to. <# of bytes> specifies how many bytes are to be moved. For example, to move 256 bytes from address 2048 to address 2816, enter:

```
Ready 2048 2816 256 MOVMEM
```

MOVMEM can be handy for relocating character sets and 3-D images in memory, as will be discussed in Chapters 7 and 8.

Retrieving Word Addresses

The word ' (an apostrophe, also called a "tic") places on the stack the address of the word that follows it, and prevents that word from being executed. Here is an example:

```
Ready ' ERASE
[30749]
```

The tic placed the address of the word ERASE on the stack, and prevented ERASE from being executed. Note that the tic is a word that looks forward down the input line, and retrieves the address when it is compiled, not every time it is executed.

The address returned by "tic" is always greater than the hexadecimal address shown with $LIST. This is because the $LIST address indicates the beginning of the word definition, and "tic" returns the address of the executing portion of the word. See Appendix B for more information on the word library structure.

Calling Machine Language Routines

Machine language programs in memory can be called directly from GraFORTH with the word CALL. CALL removes a number from the stack, interprets it as a memory address, then calls the machine language routine at that address. (The routine should end with an RTS (ReTurn from Subroutine) instruction to return to GraFORTH properly.) Machine language programs can be loaded from disk using the DOS command "BLOAD" into any free area of memory, then CALLed from GraFORTH.

Before a machine language CALL is made, values can be placed in the Apple processor's A, X, Y and P registers using the GraFORTH variables AREG, XREG, YREG and PREG. Before making the machine language CALL, simply place the desired values into AREG, XREG, YREG and PREG as you would any other variable. When CALL is executed, it loads the processor registers with the values from these variables before doing the call. (Note the importance of loading a proper value into PREG. If improper processor bits are set, GraFORTH will not operate!) After the routine has executed, the values of the registers are loaded back into the variables and can be read from GraFORTH, again, just as any other variable.

Here is a nice example, which uses CALL to read the game paddles. The Apple System monitor contains a routine at location -1250 for reading the game paddles. It expects to see the number of the game paddle (0 to 3) in the processor's X register. It returns a number from 0 to 255 (based on the position of the paddle) in the Y register. The following word reads the value of a game paddle by placing the top stack value in XREG, calling the paddle routine, then placing the value of YREG on the stack:

```
: READ.PADDLE
  -> XREG
  -1250 CALL
  YREG ;
```

(The Apple manuals warn that two consecutive readings of a game paddle can produce incorrect results, and suggest a short wait loop between readings.)

Compiling Number Tables

The word "," (comma) causes a number to be compiled as a byte directly into GraFORTH. Small assembly language routines can be compiled using commas, or number tables can be generated. Here is an example of a word that contains a number table of the visible high resolution colors. The numbers are stored as individual bytes following the word name in memory:

```
: COLOR.TABLE 1 , 2 , 3 , 5 , 6 , ;
```

These numbers correspond to the colors green, violet, white, orange, and blue. (Colors in GraFORTH will be discussed in detail in the next chapter.) Each number can be accessed by using the tic to retrieve the address of COLOR.TABLE, then adding an offset (0 to 4) to pick out the appropriate number with PEEK. Note that COLOR.TABLE is not an executable word!

The comma is the only GraFORTH word that assembles directly at the byte level, and some precautions are required to use it effectively. The comma should only be used within word definitions. Also, for internal reasons, the first byte of an assembly of code or data may not be greater than 127 (hexadecimal $7F), nor can it be equal to 10 ($A). Here are the reasons: 10 is a special reserved compiler flag, and a number less than 128 must follow each GraFORTH word name to mark its end. (For more information, see Appendix B for technical information on GraFORTH's dictionary link structure.)

Leaving GraFORTH (gently)

The GraFORTH word "BYE" can be used to enter the Apple][system monitor. The GraFORTH language begins at hex location $6000. To restart GraFORTH from the monitor, type "6000G".

Conclusion

That about wraps up the language features of GraFORTH. From here on out we'll be talking about the many types of graphics available with GraFORTH. (That is what you bought it for, isn't it?) The next chapter will cover basic point and line drawing in GraFORTH, as well as a discussion of the supplied TURTLEGRAPHICS. We'll get into the various modes, color selections and...

Well, that's the topic of chapter 6!

CHAPTER SIX: TWO-DIMENSIONAL GRAPHICS

Chapter Table of Contents: Page

Purpose and Overview

The graphics capabilities of GraFORTH can be divided into three main groups:

Two-Dimensional Graphics (or "Graphics of the First Kind") includes commands that plot points, draw lines, and fill rectangular areas on the screen, using a variety of colors and options.

Character Graphics (or "Graphics of the Second Kind") includes using and creating new character sets, displaying text with different sizes and colors, and defining completely new shapes and pictures in terms of character sets and displaying these shapes using a special block printing function.

Three-Dimensional Graphics (or "Graphics of the Third Kind") includes creating and displaying three-dimensional color images at high speed for animated effects.

This chapter will discuss two-dimensional graphics. We'll start by talking about what the Apple itself is capable of, and how GraFORTH uses these capabilities. We'll show you how to plot points and draw lines, and then undraw them again, effectively removing them from the screen. We'll discuss color and the drawing modes (ORMODE and EXMODE) and how they affect the drawing process. We'll also talk about using Turtlegraphics, which is especially useful for creating certain kinds of graphics displays.

Apple Graphics

The Apple screen display, whether it be text or graphics, is made out of the same units, called pixels. A pixel (abbreviated form of 'picture cell') is the smallest unit, or dot, which may be turned on or off of the surface or the screen. There are 53,760 of these smallest units which make up the entire screen, arranged in a matrix 280 dots wide and 192 dots high.

The standard Apple text display divides the screen into 24 horizontal lines, each 8 dots high. Seven of these 8 vertical dots are used to form the characters, while the eighth is used to separate the lines from one another. Horizontally, the screen is divided into 40 columns, each 7 dots wide. Five of these 7 horizontal dots are used to form the character, while one on each side of the character is used for spacing between the characters. The ASCII values for the characters on the text screen are stored in a 1024 byte memory area. The hardware inside the Apple continuously reads the values from this area and places the appropriate characters on the screen.

The Apple graphics display allows you to turn on or off all 53,760 dots on the screen individually. There are two 'graphics pages' in memory reserved for this function, but because of the higher resolution, each requires 8192 bytes to be set aside. It is possible to alternate between the pages very rapidly for animation effects (GraFORTH does this automatically for 3-D displays), but the Apple display hardware cannot merge or blend the information on the two pages. These two high resolution pages are often called 'picture buffers'. Each dot on the screen represents one bit from the picture buffer. Seven of the 8 bits in each byte are displayed on the screen, with the last bit used in determining the colors of the other dots in that byte.

GraFORTH Graphics

While it is possible to use the Apple text display from GraFORTH (with the word TEXT), the usual display is the graphics display. To specify points on the graphics screen, GraFORTH uses 'Cartesian coordinates'. This is a straightforward way to select a point by naming the column and the row the point is in. The horizontal position is the X coordinate and the vertical position is the Y coordinate.

The range of screen coordinates for GraFORTH graphics is:

X from 0 (screen left) to 255 (screen right)

Y from 0 (screen top) to 191 (screen bottom)

Thus, the upper-left corner of the screen can be represented with X=0 and Y=0, or simply the X-Y pair (0,0).

Note: The GraFORTH graphics screen is 9 percent narrower than the maximum possible (256 points wide rather than 280) for the sake of operating speed. This is one factor that contributes to GraFORTH's fast line drawing.

The standard Apple text display still uses all 280 dots across the screen for 40 characters per line. The characters themselves, instead of being placed on a text screen by the Apple hardware, are "drawn" from the text page onto the graphics picture buffer. The full character space, 7 dots by 8 dots, can be used, and is used for lower case characters and special character styles.

Two-Dimensional Graphics Words

PLOT, LINE and FILL

For these examples, we don't want text scrolling all over our beautiful graphics, so let's establish a text window in the bottom part of the screen. These examples will keep the graphics above the text window and away from harm. To establish the window, type:

```
Ready 0 40 18 24 WINDOW
```


This sets a 40-column wide window from line 18 to the bottom of the screen. Now type:

```
Ready ERASE
```

This clears the text that was still above the text window.

Let's begin at the beginning, with plotting points. The GraFORTH word PLOT removes two numbers from the stack, interprets them as X and Y coordinates, and plots a point at those coordinates on the screen. The form for PLOT is:

<X-coordinate> <Y-coordinate> PLOT

This example will plot a point in the upper left corner of the screen:

```
Ready 0 0 PLOT
```

Here is another point, in the upper right portion of the screen:

```
Ready 200 25 PLOT
```

The word LINE, like PLOT, removes two numbers from the stack and interprets them as X and Y coordinates. LINE then draws a straight line from the last plotted point to the given coordinates. To draw a line, we use the last point we plotted as one of the endpoints. We simply give LINE the coordinates of the other endpoint:

```
Ready 50 100 LINE
```

This draws a diagonal line from the point (200,25) to (50,100). We can draw another line, by using PLOT and LINE together again:

```
Ready 100 10 PLOT  100 140 LINE
```

This draws a vertical line through the other line and almost into our text window.

Rectangular areas can be filled in quickly with the word FILL. FILL also removes X and Y coordinates from the stack. It treats the last plotted point as one corner of the area, and the given coordinates as the opposite corner. This example fills in a rectangular area on the right side of the screen:

```
Ready 120 125 PLOT

Ready 200 75 FILL
```

For both LINE and FILL, the "last plotted point" is always the point last used by a plotting word, whether it was PLOT, LINE, or FILL. Another word, POSN, removes X and Y coordinates from the stack to act as a "last plotted point" without doing any plotting. POSN can be used to determine the first endpoint of a line or one corner of an area. This example uses POSN to set the first endpoint of a line:

```
Ready 225 50 POSN

Ready 250 125 LINE
```

COLOR

Of course, GraFORTH can draw in colors, too! The color is set with the word COLOR. COLOR removes a number from the stack and uses it to select a color. The eight color numbers (0 through 7) are the same as those used by Applesoft Basic. Here is a listing of the graphics colors:

Color Number	Color		
0	not used		
1	Green	(1)	
2	Violet	(1)	
3	White	(1)	
4	not used		
5	Orange	(2)	(depends on monitor)
6	Blue	(2)	(depends on monitor)
7	White	(2)	

The orange and blue colors may appear different shades on different color monitors. The colors can be divided into two groups. The numbers in parentheses represent the "group number" (either 1 or 2). Because of some Apple][hardware constraints, it may be desirable to use colors from the same group when drawing lines or areas close together. We'll show you an example of this in a bit. (The Apple][Reference Manual contains more information on the internal details of these constraints.)

If you don't mind a bit of typing, this example will display 6 diagonal lines in each of the visible colors:


```
Ready ERASE

Ready 1 COLOR 0 0 PLOT 100 100 LINE

Ready 2 COLOR 20 0 PLOT 120 100 LINE

Ready 3 COLOR 40 0 PLOT 140 100 LINE

Ready 5 COLOR 60 0 PLOT 160 100 LINE

Ready 7 COLOR 100 0 PLOT 200 100 LINE
```

With your color monitor properly adjusted, the colors of these lines (from left to right) should be green, violet, white, orange, blue, and another brand of white. Note that the colored lines are not broken at all, as they are with some graphics displays (like Applesoft). GraFORTH draws all colored lines without breaks.

Lines and points can be drawn over FILLed areas, but the colors will be affected:

```
Ready ERASE

Ready 5 COLOR

Ready 0 0 PLOT 100 100 FILL
```

This draws an orange rectangle in the upper left portion of the screen. Now let's draw a line of a different color through it:

```
Ready 6 COLOR

Ready 0 0 PLOT 100 100 LINE
```

Note that 6 COLOR specifies blue, but because of the orange background, the line appears white. Now let's try the same example again, this time using colors from different color groups:

```
Ready ERASE. 5 COLOR

Ready 0 0 PLOT 100 100 FILL

Ready 1 COLOR

Ready 0 0 PLOT 100 100 LINE
```

Whoops! You should see a series of small green rectangles along the diagonal. This is the result of the Apple][hardware limitations. The solution to avoiding this trouble is to simply use colors of the same group when lines or areas are superimposed or placed close together.

UNPLOT, UNLINE, and EMPTY

So far we've been using the word ERASE to clear the graphics from the screen. In GraFORTH, points, lines, and areas can be selectively erased. Let's ERASE the entire screen now and set the color back to white, then plot a few points:

```
Ready ERASE  3 COLOR

Ready 50 25 PLOT

Ready 100 25 PLOT

Ready 150 25 PLOT
```

Points can be individually removed with the word UNPLOT. UNPLOT has the same form as PLOT, however it erases the point at the given coordinates. (If there is no point there to begin with, nothing happens.) Let's use UNPLOT to erase two of the points we have on the screen:

```
Ready 50 25 UNPLOT

Ready 100 25 UNPLOT
```

Similarly, lines can be erased with the word UNLINE. This example draws two lines, then erases one of them:

```
Ready 0 0 PLOT 100 100 LINE

Ready 50 0 PLOT 150 100 LINE

Ready 0 0 UNPLOT 100 100 UNLINE
```

Rectangular areas created with FILL can be erased with the word EMPTY. Here we'll FILL two areas, and erase one:


```
Ready 25 75 PLOT 100 125 FILL

Ready 175 25 PLOT 225 100 FILL

Ready 25 75 UNPLOT 100 125 EMPTY
```

Points, lines, and areas must be UNdrawn using the same color they were drawn in. For example, all of the above objects were drawn with 3 COLOR set. The same color was still in effect when some of the objects were erased. Let's change the color and try erasing the remaining line and area:

```
Ready 1 COLOR

Ready 50 0 UNPLOT 150 100 UNLINE
```

Since 1 COLOR is set, the GraFORTH system assumes a green line is to be erased, and leaves a string of violet dots behind.

```
Ready 2 COLOR

Ready 175 25 UNPLOT 225 100 EMPTY
```

With 2 COLOR set, GraFORTH tries to erase a violet colored area, changing the white to green.

INVERSE and NORMAL

If you prefer to do graphics on a white background, you can do this with the word INVERSE. INVERSE simply draws the 'complements' of the selected color: white becomes black, black becomes white, green becomes violet, blue becomes orange, etc. To show the effects of INVERSE, let's first erase the screen, then enter INVERSE:

```
Ready ERASE

Ready INVERSE
```

Notice that the "Ready" on the last line is now displayed in "inverse": black characters on a white background. Since only the word "Ready" was printed after executing INVERSE, it is the only thing displayed in inverse. Now type:

```
Ready HOME
```

Since HOME clears the text window, now everything inside the text window is in inverse. Now type:

```
Ready ERASE
```

ERASE has "erased" the entire screen to white. Let's draw the six colored lines again:

```
Ready 1 COLOR 0 0 PLOT 100 100 LINE

Ready 2 COLOR 20 0 PLOT 120 100 LINE

Ready 3 COLOR 40 0 PLOT 140 100 LINE

Ready 5 COLOR 60 0 PLOT 160 100 LINE

Ready 6 COLOR 80 0 PLOT 180 100 LINE

Ready 7 COLOR 100 0 PLOT 200 100 LINE
```

Note that the colors of the lines have all changed. From left to right, the colors are now violet, green, black, blue, orange, and another black.

We'll eventually want to return to a normal black-background display. The word NORMAL causes GraFORTH to use the normal colors again, including good ol' black:

```
Ready NORMAL

Ready ERASE
```

ORMODE and EXMODE

GraFORTH has two different "drawing modes", called "ORMODE" and "EXMODE". Amazingly enough, these modes are set with the GraFORTH words ORMODE and EXMODE. The 'default' mode (the mode GraFORTH uses when a mode is not specified) is ORMODE. The philosophy behind ORMODE is that the plotting words put dots of the specified color on the screen regardless of what is already on the screen. With EXMODE however, a drawing command will put points on the screen only where points are not already plotted. If some points to be plotted are already plotted, those points will instead be turned off.

A couple of examples will be helpful here. Let's first FILL an area, then draw an overlapping line in ORMODE:

```
Ready 100 50 POSN  150 100 FILL

Ready 50 50 POSN  200 100 LINE
```

The line goes straight through the middle of the rectangle. Watch what happens when we try to erase the line:

```
Ready 50 50 POSN 200 100 UNLINE
```

The line was erased, but it neatly chopped the rectangle in half, too. Using EXMODE, anything that can be done can also be undone. Let's do the same example again, this time in EXMODE:

```
Ready ERASE  EXMODE

Ready 100 50 POSN 150 100 FILL

Ready 50 50 POSN  200 100 LINE
```

The line is white, except where it passes over the white background of the rectangle. Here it is changed to black. Now to erase the line, we want to make the white sections black, and the black trace through the rectangle white. And this is exactly what happens with regular plotting in EXMODE. We can erase the line by telling GraFORTH to draw it again:

```
Ready 50 50 POSN 200 100 LINE
```

The line is erased, and the rectangle is again intact. The key to understanding EXMODE is that if something is drawn once, it appears on the screen. If it is drawn again, it disappears, leaving the screen as if the object had never been drawn.

EXMODE works equally well with colors. In this example, a green line is drawn through the rectangle, the white rectangle is erased, then the line is erased:

```
Ready 1 COLOR 50 50 POSN 200 100 LINE
```

Notice that the line is violet inside the rectangle.

```
Ready 3 COLOR 100 50 POSN 150 100 FILL
```

The line is now completely green, as if the rectangle never existed.

```
Ready 1 COLOR 50 50 POSN 200 100 LINE
```

EXMODE and ORMODE can be combined with INVERSE and NORMAL along with the six colors to produce a wide variety of color and pattern combinations, more than we could hope to fully explore here. We suggest that you experiment further with these various combinations, to see how they can work best for your applications.

GPEEK

Your programs can determine whether or not a given point on the screen has been plotted with the word GPEEK. GPEEK removes X and Y coordinates from the stack, looks to those coordinates on the screen, and places a non zero number stack if the point there is "on" (not black) or a zero if the point is "off" (black). The following example draws a line, then checks two points, one on the line and one off:

```
Ready 3 COLOR 0 0 PLOT 100 100 LINE

Ready 50 50 GPEEK .
?

Ready 200 10 GPEEK.
0
```

Turtlegraphics

Turtlegraphics is also available from GraFORTH. Turtlegraphics is a somewhat different way of specifying how to draw lines in GraFORTH. Imagine a tiny turtle sitting on the middle of the screen with ink on his tail. Wherever he moves he draws a line behind him. We can tell him to turn to the left or the right, and we can tell him to walk forward a given distance leaving a straight line behind him. (For the mathematicians among us, this way of drawing lines could be considered as using "relative polar coordinates".)

The Turtlegraphics words in GraFORTH are found on the system disk in a text file called "TURTLE". We can compile these words into the dictionary by typing:


```
Ready READ " TURTLE "
```

We can see the words added to the dictionary by typing LIST. A few of the words are used by the other words: TURTLE.X, TURTLE.Y, and TURTLE.ANG are variables, and TURTLE.WALK is called by both MOVE and MOVETO.

Let's "initialize" Turtlegraphics by typing:

```
Ready TURTLE
```

TURTLE resets graphics mode, erases the screen and sets a text window along the bottom four lines, then sets 3 COLOR (white) and positions the turtle in the center of the screen, facing toward the top.

MOVE

The word MOVE moves the turtle in the direction it is pointing, drawing a line. The form is:

```
<distance>  MOVE
```

The distance is measured in pixels, or dots. To move the turtle 50 pixels, type:

```
Ready 50 MOVE
```

TURNTO

The turtle can be turned to a certain angle with TURNTO. TURNTO has the form:

```
<angle>  TURNTO
```

The angle given is in degrees, and increasing angles are in a clockwise direction. Zero is straight up, 90 is to the right, 180 is facing down, and 270 is to the left. Let's move the turtle in our example to face to the right (to 90 degrees), then move it 75 pixels:

```
Ready 90 TURNTO

Ready 75 MOVE
```

TURN

The word TURN turns the turtle clockwise from its current direction a given angle. The form is the same as for TURNTO, but TURN is a relative turn from the turtle's current direction. The following example now turns the turtle 45 more degrees clockwise, then moves the turtle 50 pixels:

```
Ready 45 TURN

Ready 50 MOVE
```

MOVETO

Lastly, MOVETO moves the turtle directly to a specified X,Y position on the screen without drawing any line. The form for MOVETO is:

```
<X coordinate>  <Y coordinate>  MOVETO
```

MOVETO is similar to POSN in that it simply establishes a new point on the screen, but MOVETO also updates the turtle's position for further Turtlegraphics commands. We can move the turtle to the upper-left corner of the screen, turn it to face to the lower-right, then move it back to the center, drawing a line, with the following commands:

```
Ready 0 0 MOVETO

Ready 127 TURNTO

Ready 160 MOVE
```

Examples

The advantage of Turtlegraphics is that shapes can be drawn in different sizes and facing different directions with little work. For example, to draw a square, you can type the following:

```
Ready TURTLE

Ready 50 MOVE 90 TURN 50 MOVE 90 TURN

Ready 50 MOVE 90 TURN 50 MOVE
```


A faster way is to repeat the words in a loop:

```
Ready TURTLE

Ready 4 0 DO 50 MOVE 90 TURN LOOP
```

This line can be put into a word definition and used at any time:

```
: SQUARE
  4 0 DO
      50 MOVE
      90 TURN
  LOOP ;
```

Now the square can be drawn starting at any point on the screen and turned any direction:

```
Ready TURTLE

Ready 0 100 MOVETO SQUARE

Ready 55 100 MOVETO 30 TURNTO SQUARE

Ready 120 100 MOVETO 60 TURNTO SQUARE

Ready 190 100 MOVETO 90 TURNTO SQUARE
```

(Note: The GraFORTH word SIN is used to compute sines of angles used in Turtlegraphics. If you have an applications program that uses angles, the word SIN can be very helpful. SIN removes a number from the stack and uses it to select and return a scaled sine value. The table repeats for every 128 numbers, and returned values range from -128 to 127.)

CHAPTER SEVEN: CHARACTER GRAPHICS

Chapter Table of Contents:

Purpose and Overview

GraFORTH can do weird and wonderful things with the characters displayed on the screen. Text can be reverse scrolled, down the screen. Characters can be made much larger, and displayed in color. Different character styles, or 'fonts' can be selected and even created in GraFORTH. Entire images can be defined within a character font and rapidly printed as a block of "characters" for animated displays.

In this chapter we'll show you how to make use of each of these features and give you some suggestions for incorporating them into your own programs.

Special Output Characters

Besides the special input characters (ConTRoL-I, ConTRoL-O, etc.) discussed in Chapter 4, GraFORTH also uses two special output characters, ConTRoL-L, and ConTRoL-K. These characters are usually printed from within a program, instead of entered at the keyboard. (They can be typed from the keyboard, but GraFORTH will try to read them as characters in a GraFORTH word.)

ConTRoL-L (Apple ASCII number 140) erases the screen inside the text window. Printing a ConTRoL-L is equivalent to executing the word HOME.

ConTRoL-K (Apple ASCII number 139) causes a reverse line feed, so that subsequent printing will be one line higher. If printing is already on the top line of the text window (the vertical tab equals the top window margin), then the display will scroll in reverse, moving text down the screen.

Changing Character Size and Color

GraFORTH has the unique ability to print characters in 8 different sizes using the word CHRSIZE. CHRSIZE removes a number from the stack to select the character size. Valid numbers are from 0 to 8. Character size 0 specifies the usual GraFORTH character display. Character sizes 1 through 8 cause the characters to be "drawn" onto the screen using GraFORTH's color graphics capabilities. Character size 1 is the same size as character size 0, and the others are 2 through 8 times larger.

Let's introduce some of these features through examples. First, we'll set everything back to normal by typing:

```
Ready ABORT
```

Now let's erase the normal sized characters from the screen and select a larger character size:

```
Ready HOME 2 CURSIZE
```

(Erasing the screen with HOME is a normal but not required step in changing character size. If HOME is not used before changing size, in some cases not all entered characters will be printed.)

The "Ready" prompt is now twice its normal size! You will notice that the large character sizes take a longer time to print, and that if allowed, scrolling is much slower than it is when using the standard character size. Also, the screen is actually 9% narrower than the standard size, since the graphics features are used to print them.

The large characters can also be displayed in color! Type:

```
Ready HOME 1 COLOR
```

This will clear the screen, then make the text green. We cleared the screen again because combining two colors of text on the screen can have some unusual effects of its own. To see these effects, type:

```
Ready 2 COLOR
```

Now hit the <return> key a few times to cause the text to scroll. The "Ready" prompt that was green gets overwritten with the violet, but does not scroll. Only text of the current color and of the current size will behave as expected with text commands.

Obviously, when the characters are larger, fewer characters can be displayed on the screen. When you select a new character size with CHRSIZE, GraFORTH automatically sets the text window size to the correct limits, to keep the text on the screen. Below is a table relating character sizes to the number of characters that can be displayed, and indicating whether or not colored text is possible for that character size:

Size	Columns	Rows	Color?
0	40	24	No
1	32	24	Yes (with funny effects)
2	16	12	Yes (with better effects)
3	10	8	Yes
4	8	6	Yes
5	6	4	Yes
6	5	4	Yes
7	4	3	Yes
8	4	3	Yes

You might want to try the following to see GraFORTH's largest character size in color. First type ABORT to get yourself back to a predictable place, then type:

```
Ready HOME  8 CHRSIZE  5 COLOR
```

A mammoth orange "Ready" prompt will appear, split across two lines, with a huge lumbering cursor! Allowing time for the text to scroll, now enter:

```
Ready INVERSE
```

After another scroll, the display changes to inverse. Obviously, you wouldn't want to enter a long program this way! Large character sizes work very well for program or game displays, but weren't really intended to be used for input. The fastest way out of our current situation (besides hitting <reset>) is to type:

```
Ready ABORT
```

After the text scrolls once more, the ABORT is executed, and things are back to normal.

Font Selection

The character "style" used in a text display (the actual set of shapes of the characters displayed) is called a character 'font', or character set. The Apple][contains an uppercase-only character set stored in its hardware. GraFORTH uses this when TEXT mode is selected. However, GraFORTH's usual graphics display instead uses a character set from memory. This character set is stored in a binary file on the GraFORTH system diskette, and is read into memory when GraFORTH is first booted.

The disk actually contains several character sets, and any of them can be used for text display. The character set files on disk are:

```
CHR.SYS
CHR.STOP
CHR.SLANT
CHR.GOTHIC
CHR.BYTE
CHR.STUFF
CHR.MAXWELL
```

(The last two are special character sets used for 'character graphics', and do not work well for a text display. We'll show you how to work with these in a bit...)

In memory, a character set occupies 768 bytes. There are 96 printable characters, and each character uses 8 bytes in the character set. These 8-byte blocks are actually graphics "pictures" of each character. When GraFORTH is booted, it loads CHR.SYS into memory starting at location 2048. Whenever it displays a character, it looks up the "picture" of that character from this area of memory, and places it on the screen.

Character sets elsewhere in memory can also be used for the screen display. Let's load another character set from disk into a free area of memory. The location 2816 is the beginning of a large free area of memory. We'll use a standard DOS call to load the file in:

```
Ready CR 132 PUTC PRINT " BLOAD CHR.BYTE,A2816 " CR
```

The disk whirs a bit, and the character set is loaded. To use this character set for the display, the word CHRADR is used. CHRADR stands for CHARacter ADdRess, and it is used to select the memory location of the current character set. The form is:

```
<address of character set>  CHRADR
```

We loaded the character set into memory starting at location 2816, so this is the address we give to CHRADR:

```
Ready 2816 CHRADR
```

All printing will now use the new character set. The characters that were already on the screen in the old character set, however, are unchanged. Characters from different character sets can be displayed on the screen at the same time. However, if the screen is scrolled, these characters will be reprinted a line higher, using the newest character set.

The ASCII numbers for the printing characters range from 160 to 255. To display all of the printing characters in the set at once using PUTC, type:

```
Ready 256 160 DO I PUTC LOOP
```

You may want to load the other character sets into memory to see what they look like. You can load them into the same area of memory and overwrite CHR.BYTE, or you can use another free area of memory and select it with CHRADR. The memory map in Appendix B shows the free areas of memory. Therefore, it is possible (and easy!) to have several character sets in memory at once, quickly changing from one to another. Care should be taken, however, to avoid overwriting a portion of the GraFORTH system. Remember that each character set occupies 768 bytes of memory.

Usually, you will want to return to the system (CHR.SYS) character set. The GraFORTH word CHRSET returns the address of this character set, 2048. Thus, to switch back to this display, you can type:

```
Ready CHRSET CHRADR
```

(Of course if you want to, you can overwrite this area of memory with another character set, too.)

The CHAREDITOR

On the GraFORTH system diskette is a file called CHAREDITOR. This program enables you to read in character sets, examine and modify character shapes, create large block images that are stored as a series of characters, and save the new character sets to disk again.

CHAREDITOR is one of the larger programs, so it would be a good idea to LIST the dictionary and FORGET any words you may have added before loading in CHAREDITOR. To load the program in, type:

```
Ready READ " CHAREDITOR "
```

To run CHAREDITOR, type:

```
Ready HOME RUN
```

Notice that we cleared the screen before running the program. CHAREDITOR does not automatically clear the screen. This is so that any graphics images on the screen can be retained and used within the CHAREDITOR, allowing you to "pull" images and shapes from other programs into your GraFORTH character sets.

You will see a list of commands to the right, the prompt "Enter command:" near the bottom of the screen, and a flashing dot in the upper-left corner. This flashing dot is the "drawing cursor" and will be used for creating your own character shapes.

Selecting and Displaying the Character Set

The character editor works with one character set at a time. To get an understanding of things, let's start by looking at the system character set that starts at location 2048. The editor uses single-letter commands. To specify the address of the desired character set, press "A" for Address. You will then see the prompt:

```
Enter Charset
Work Area Address : 2816
```

The input cursor is flashing over the "2816". This is the default address, the address used if you do not specify one. You can keep this address simply by pressing <return>. However, we want to enter the address of GraFORTH's standard character set. Type "204A" over the top of the "2816" and press <return>. Now 204A is the address of the character set used by the character editor.

Type "D" for "Display characters". You'll see a display across the bottom of the screen of all the characters in the character set, in inverse. To the left are the numbers 0, 32, and 64. These are index numbers. When manipulating character shapes in GraFORTH, character numbers in the range of 0 to 95 are used instead of the ASCII values (which range from 160 to 255 for printing characters). The first row of characters are numbered 0 through 31, the second row 32 through 63, and the third row 64 through 95.

Displaying a Block of Characters

If we want, we can take a sequential string of characters and display them in a rectangular block on the screen. Let's display the 6 characters "n" through "s" in a block that is 3 characters wide by 2 characters tall. To select a block of this size, press "B" for "Blocksize". You will be prompted:

```
Enter Block Horizontal Size :
```

Enter a 3 and press <return>. You will see:

```
Enter Block Vertical Size :
```

Enter a 2, press <return>, and you will get the regular "Enter command:" prompt back. Also notice that 4 more dots have appeared at the top of the screen, outlining our 3 by 2 character block.

Press "D" to bring the character set display back. Counting across the bottom row from the index number 64, you will find that the character "n" is character number 78. To display the block of 6 characters starting with "n", type "R" for "Read".

You will see:

```
Enter character number
to be read :
```

We want character number 78, so type "78". The 6 characters will appear in the block surrounded by the 4 dots.

You can also display blocks starting on other characters, or use a different blocksize. When changing blocksize, you may want to erase the block from the screen. To do this, simply type "E" for "Erase", then answer "Erase (Y/N) :" with a "Y".

We've been looking at a block of standard characters, to show you how block printing is done. Now let's see some actual character graphics. To protect our precious system character set, press "A" and select an address of 2816 again, back into open memory. Type "G" for "Get". This option allows you to load a character set in from disk. You will see:

```
Enter Load File Name :
```

Type "CHR.STUFF".' This character set will load into memory starting at the location 2816. Type "D" to display this character set. Except for a few punctuation symbols, those don't look much like characters! You can see pieces of the Insoft logo, parts of faces, and an assortment of lines which are actually pieces of a helicopter used in the GraFORTH demonstration program.

If you've changed the Blocksize, set it back to 3 characters horizontally by 2 characters vertically. Now type "R" and read character number 78. A smiling face will appear in the upper left. By pressing "D" again, you can see that this face occupies the same six characters that the characters "n" through "s" occupied in the system character set. The other three faces begin at character numbers 84 and 90. Just press "R" and enter the character number to see them.

The Insoft logo uses a blocksize of 8 by 2 characters, and begins at character number 16. The three helicopters use a blocksize of 5 by 3 characters and begin at character numbers 33, 48, and 63. You will probably want to erase the block (with "E") before changing the blocksize, so that part of the previous image won't remain on the screen beside the new block.

Defining Your Own Shapes

To create your own shapes with the character editor, first select a blocksize for the image you want to draw. Erase the block if necessary. Here's where the drawing cursor comes in. By pressing the I, J, K, and M keys, you can move this cursor one pixel up, left, right, or down within the block. If you want to plot a point at the position of the cursor, press "P" for "Plot". To draw a line from the last plotted point to the cursor, press "L". Notice that "P" and "L" are actually PLOT and LINE commands, with the coordinates specified by the cursor. The character image is created by moving the cursor and drawing the points and lines that make up the image.

In addition, you can create character images in color. Press "C" for "Color" and enter the number of the color you want to work in. (When colored character images are displayed in GraFORTH, the colors may be different, depending on whether the image is drawn beginning on an odd-numbered column or an even-numbered column. This comes about as a result of the way the Apple][generates high-resolution color.)

If you plot a point that you didn't want, you can erase it by pressing "U",which UNPLOTs the point. Similarly, you can erase lines by pressing "Z". If the drawing cursor moves too slowly, you can increase its step size by pressing "X", then entering the number of pixels you want the cursor to move whenever you press a cursor-moving key (I, J, K, M). If your image isn't coming out the way you'd like....well, press "E" to erase it and try again!

Experiment with these keys to get a feel for creating images. All of the images in CHR.STUFF were created with the character editor. If you like, you can read an existing image from the character set and use the drawing keys to modify it.

When you've created an image that you want to save, first multiply the block vertical size by the horizontal size, to determine how many characters your image will occupy. Then press "D" to see the current character set, and choose a range of characters in the character set to write your image to. Press "W" for "Write". You will be prompted:

```
Enter character number
to be written :
```


Type the character number of the first character in the desired range. Your image will be written into the character set starting at that character. Press "D" again and you will see your image neatly dissected and placed in the character set.

Images from one character set can be copied to another using the CHAREDITOR "T" ("Transfer") option. You will be prompted for a "From" address, a "To" address, and a length. To copy an entire character set from one address to another, simply enter the address of the character set to be transferred, the address of where it is to go, and enter 768 for the length. Remember that character sets are 768 bytes long.

Transferring only part of a character set is a little trickier. Remember that each character occupies 8 bytes. Compute the "From" and "To" addresses based on the character number and the addresses of the character sets. The length is the number of characters times 8.

Saving a Character Set

After a new character set has been created, you can save it to disk to be used again later. To save a character set, press "S" for "Save". You will see:

```
Enter Save File Name :
```

Type the filename you've selected for the character set. Be sure that there are no files with that name on disk, unless you want to overwrite that file. Note that all of the character sets on the GraFORTH system disk begin with the prefix "CHR.". This is not a requirement; the prefix simply acts as a reminder that the file contains a character set.

When you want to leave the character editor, type "Q" for "Quit". If you want to begin work with another program, it would probably be best to FORGET the character editor first, since it takes up a lot of room in the word library. The word "X" is the first word in the character editor, so to delete the editor, type:

```
Ready FORGET X
```

Block Printing from GraFORTH

Printing blocks of characters is done directly from GraFORTH much the same way as in the character editor. A character set is loaded into memory, an appropriate blocksize is selected, and a sequential range of characters is printed in the block at the current horizontal and vertical position.

Let's display some of the same images we saw earlier in the character editor. First, load "CHR.STUFF" back into memory:

```
Ready CR 132 PUTC PRINT " BLOAD CHR.STUFF,A2816 " CR
```

You could now type "2816 CHRADR" to select the character set, but remember that this character set doesn't have much in the way of recognizable characters! It contains helicopter parts and other things. GraFORTH can recognize the characters fine, but the screen display is unusable. When we display a character image, we'll jump into the character set, display the image, then jump back out.

BLKSIZE

The block size in GraFORTH is set with the word BLKSIZE. The form for BLKSIZE is:

```
<horizontal size>  <vertical size>  BLKSIZE
```

As in the character editor, the horizontal and vertical size are measured in characters. BLKSIZE remains set until changed. The word ABORT does not reset BLKSIZE.

To prepare to see the smiling faces, set a blocksize of 3 characters wide by 2 characters tall:

```
Ready 3 2 BLKSIZE
```


PUTBLK

The word that actually puts the block of characters on the screen is PUTBLK. PUTBLK removes a number from the stack and uses it as the starting character number for the block to be displayed. Character numbers range from 0 to 95, as in the editor. The number of characters to be printed is determined by BLKSIZE. The position of the block on the screen is set the same way text is positioned, with HTAB and VTAB, or the other text positioning commands.

Let's block-print one of the faces in CHR.STUFF. For this example, type this entire line at once:

```
Ready HOME  2816 CHRADR  78 PUTBLK  CHRSET CHRADR  12 VTAB
```

"HOME" clears the screen and positions printing to the upper-left corner, "2816 CHRADR" sets the character set address for CHR.STUFF, "78 PUTBLK" actually prints the image, "CHRSET CHRADR" resets the system character set, and "12 VTAB" gets the following "Ready" prompt down out of the way, so that it won't overwrite the block just printed.

A smiling face should have appeared in the upper-left corner of the screen.

To save on typing a bit, let's define a couple of new words to help us in and out of the special character set. We'll call these words "IN" and "OUT":

```
Ready : IN  2816 CHRADR  HOME ;

Ready : OUT  CHRSET CHRADR  12 VTAB ;
```

To display another face, we can simply type:

```
Ready IN 84 PUTBLK OUT
```

Unlike text printing, PUTBLK does not update the horizontal cursor position. Therefore, once a printing position has been established, several images can be drawn sequentially in the same space. The following example prints the three helicopter images in the same space 100 times. Keep your eyes open; it's fast:

```
Ready 5 3 BLKSIZE

Ready IN 100 0 DO 33 PUTBLK 48 PUTBLK 63 PUTBLK LOOP OUT
```

After changing the blocksize, the Insoft logo (which starts at character number 16) can be displayed centered on the screen:

```
Ready 8 2 BLKSIZE

Ready IN 5 VTAB 16 HTAB 16 PUTBLK OUT
```

We're being cautious about the display here because we're mixing the printing of block images using one character set with reading keyboard input using another. Most finished programs will have the changes planned out, so that the most effective mixing of character images and text display can occur.

To erase a character image, the word UNBLK is used. UNBLK simply erases a block in the current blocksize at the current printing position. The following example erases the Insoft logo we placed on the screen:

```
Ready 5 VTAB 16 HTAB UNBLK
```

The VTAB and HTAB determine the position of the block to be erased. Since UNBLK doesn't print any characters, we don't need to specify a character set.

Of course, character images can also be made larger by using CHRSIZE. This example displays the Insoft logo four times as large:

```
Ready IN 3 CHRSIZE 1 COLOR 16 PUTBLK 0 CHRSIZE OUT
```

EXMODE Character Graphics

Character sizes 1 through 8 will be drawn in "EXMODE" if EXMODE is set. This allows you to draw characters or character images over other graphics, then erase them, leaving the original graphics intact. However, EXMODE character graphics requires a few special considerations.

As GraFORTH displays characters on the graphics screen, it stores the ASCII values for those characters in the text screen area. If a character about to be printed is already in place on the screen, no high-resolution printing is done, since the character is already present. This saves much time in printing and scrolling.

However, when using EXMODE, you usually want to reprint the same characters in the same location to cause them to disappear again. Therefore, to unprint a line using EXMODE, you must first erase the text screen (this is the actual Apple][text screen, not the high resolution screen used by GraFORTH) to force a reprinting. To do this, you use the Apple][monitor's screen erase routine ("-936 CALL"), then print the same line in the same position. The following word definition is an example of using EXMODE character graphics. It draws a diagonal line, writes text over the line, then erases the text, leaving the line intact. It repeats this 4 times:

```
: EXMODE.DEMO
  ERASE
  1 CHRSIZE                  (Set up EXMODE character graphics)
  EXMODE
  0 0 PLOT 100 100 LINE      (Draw the line to be written over)
  4 0 DO                     (Loop 4 times)
      3000 0 DO LOOP         (Delay loop, to slow it down)
      5 VTAB
      5 0 DO                 (Print the line 5 times)
         PRINT " This line can be erased " CR
      LOOP
     -936 CALL               (Erase the text screen)
  LOOP
  0 CHRSIZE ;
```

Summary

Output Characters

GraFORTH uses two special output characters: ConTRoL-L erases the screen inside the text window, and ConTRoL-K causes a reverse line feed, making the screen reverse scroll if the cursor is at the top of the text window.

Character Sizes

The GraFORTH word CHRSIZE uses a number from the stack to select a character size. Valid numbers are 0 through 8. Sizes 1 through 8 can be drawn in color using the word COLOR. Character size 0 is the normal text display.

Font Selection

Various character fonts can be used by BLOADing them into free memory and selecting that memory with CHRADR. GraFORTH's system character set begins at location 2048. The word CHRSET returns this address.

CHAREDITOR

The program CHAREDITOR is used to modify and save character shapes and images. Here is the normal sequence of events in the use of CHAREDITOR, with example entries:

1. Load and run the CHAREDITOR program:

```
Ready READ " CHAREDITOR "

Ready HOME RUN
```

2. Select a character set work address:

```
Enter Charset
Work Area Address : 2816
```

3. (optional) Load a character set:

```
Enter Load File Name : CHR.STUFF
```

4. Select a block size (single characters are always 1 by 1; images may be larger):

```
Enter Block Horizontal Size : 3
Enter Block Vertical Size : 2
```

5. Draw the image or character using the described sketching keys.

6. Write your image or character into the character set:

```
Enter Character Number to be Written : 90
```

7. Save the modified character set to disk:

```
Enter Save File Name : CHR.TEST
```


Block Printing from GraFORTH

Displaying character graphics from GraFORTH usually involves the following steps:

1. Load a character set into memory:

```
Ready CR 132 PUTC PRINT " BLOAD CHR.STUFF,A2816 " CR
```

2. Select the character set:

```
Ready 2816 CHRADR
```

3. Choose an appropriate blocksize:

```
Ready 3 2 BLKSIZE
```

4. (optional) Select a character size and color:

```
Ready 2 CHRSIZE  1 COLOR
```

5. Position the cursor and draw the block:

```
Ready 5 VTAB 2 HTAB 90 PUTBLK
```

Since PUTBLK does not advance the cursor, several blocks may be drawn on top of one another without having to reposition the cursor. The word UNBLK erases a block at the current position of the given blocksize.

EXMODE Character Graphics

Character sizes 1 through 8 may be drawn using EXMODE. This way, characters can be displayed over other graphics without erasing them. However, to erase a line printed in EXMODE, the text screen must first be erased with "-936 CALL" before the line is reprinted.

Conclusion

This chapter introduced GraFORTH's character graphics capabilities. So far we have covered the language features of GraFORTH, its point and line graphics, and now the set of graphics that manipulate characters and block images. Next chapter, we'll introduce the most amazing aspect of GraFORTH, its three dimensional color graphics capability. So hold on to your keyboard, here we go!

CHAPTER EIGHT: 3 D GRAPHICS

Chapter Table of Contents:

Purpose and Overview

Perhaps the most exciting aspect of GraFORTH is its high-speed 3-D graphics capabilities. GraFORTH can manipulate up to 16 three-dimensional shapes simultaneously. In this chapter we'll discuss how to use these features.

We'll begin with an overview of how 3-dimensional shapes are accessed and manipulated, and give you some introductory examples. We'll then explain the various 3-D parameters and discuss the image "format" in detail. We'll show you how to use the IMAGEDITOR to create your own 3-D images, then discuss 3-D display methods. Lastly, we'll discuss two very useful programs for developing and manipulating your 3-D image files.

3-D Graphics at a Glance

To display a 3-D object in GraFORTH, the "image" information describing the shape of the object is first loaded into a free area of memory, then commands are entered which tell the GraFORTH system where the image is in memory, and how the image is to be displayed.

GraFORTH uses an internal array to store the current information about all of the 3-D objects being displayed. The array stores the locations in memory of the actual images and the display parameters (position, rotation, size, etc.). A number (from 0 to 15) is used to refer to each object, and to select which object is currently being manipulated.

To view a 3-D image, let's first make sure things are back to normal:

```
Ready ABORT
```

and set a text window so that text doesn't scroll over our 3-D images:

```
Ready 0 40 20 24 WINDOW ERASE
```


Now let's load an image from disk into a free area of memory. The binary file "XYZ" on the GraFORTH disk contains an image of three arrows, each a different color, and each pointing a different direction. This is the same object that was used in the PLAY demonstration in Chapter 1.

```
Ready CR 132 PUTC PRINT " BLOAD XYZ,A2816 " CR
```

Before we can view "XYZ", we have to initialize the internal 3-D graphics array. Since we're starting from scratch, enter the word OBJERASE. OBJERASE clears the array, and should be used when beginning all 3-D programs.

```
Ready OBJERASE
```

Now we want to assign a number to the object we're about to view. Remember that GraFORTH can handle up to 16 objects at a time. The word OBJECT is used to specify which object to manipulate. OBJECT removes a number from the stack, and uses this number to select the current object. Let's give the image "XYZ" the number 0 in the array:

```
Ready 0 OBJECT
```

For our example, we will want the shape to be drawn automatically after each entered command. To do this, the word AUTODRAW is used. AUTODRAW removes a number from the stack. If this number is 1, then the currently selected object will automatically be drawn after each graphic command. If the number is 0, then automatic drawing will not occur. (Entering the word DRAW will draw the objects when AUTODRAW is not in effect.) Let's turn on automatic drawing with AUTODRAW:

```
Ready 1 AUTODRAW
```

We've initialized the array, set object number 0, and turned on automatic drawing, but we haven't specified where the current object is in memory. The word OBJADR is used to specify this address. We loaded the object into memory starting at 2816, so this is the number we give to OBJADR:

```
Ready 2816 OBJADR
```

At this point (because AUTODRAW is turned on) the image will appear on the screen. Right now it looks like a single arrow with a line through it, but that's only because we're seeing it head-on.

GraFORTH has 12 separate words for controlling the position, size, and orientation of 3-D objects. We'll introduce these words properly in a bit, but to give you a taste, let's rotate the image a little for better viewing:

```
Ready 14 YROT
```

Now it's beginning to come into view, and you can see parts of all three arrows. Let's move it a little more:

```
Ready 16 XROT
```

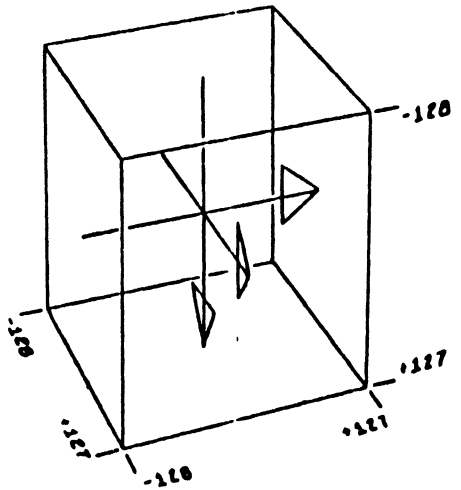
and add a little perspective:

```
Ready 6 SCALZ
```

3-D Image Format

Just as two-dimensional graphics use Cartesian coordinates labeled X and Y, three-dimensional graphics use a Cartesian coordinate system with the three directions labeled X, Y, and Z. The arrows in "XYZ" represent the three directions, or three 'axes'. X is a point along the horizontal, from left to right. Y is a point on the vertical, from top to bottom. Z is a point from rear to forward, pointing at the viewer.

The points that make up a 3-D image are expressed as three numbers, one for each of the X, Y, and Z coordinates. The valid range for each of these numbers is -128 to +127. Each arrow lies on an axis, with two coordinates equal to zero, and the ends of each arrow reaching from -128 to 127. At the center of the cube, where all three arrows meet, the three coordinates are all equal to zero.



The above diagram shows the limits for each of the three coordinates. Note that these limits define a "cube of space", 256 units along each side. All 3-D objects reside in this space. When more than one object is being displayed, each object has its own 3-D space, though these spaces may overlap or even coincide on the screen.

Image Parameters

Once an image has been loaded into memory and selected with OBJECT and OBJADR, it can be rotated, positioned, scaled, and translated in a number of ways.

Rotation

An image can be rotated around any axis, using XROT, YROT, or ZROT. XROT rotates the image around the X-axis, YROT around the Y-axis, and ZROT around the Z-axis. Each of these words removes a number from the stack and rotates the image to the selected angle. Angles are specified in units between 0 to 256 rather than degrees. An entry of 0 to YROT (or for that matter, XROT or ZROT) rotates the image around to a normal position facing the viewer. An entry of 64 rotates to 90 degrees, 128 rotates to 180 degrees, and so forth, until 256, which (like 360 degrees) is the same as zero: a full revolution.

Earlier, we used XROT and YROT to tip the image a bit so that we could get a better view. We can also use a loop and cause the image to rotate a full circle. The following word definition executes YROT repeatedly, with an increasing rotation value:

```
: YSPIN
  260 0 DO
    I YROT
  4 +LOOP ;

Ready YSPIN
```

When YSPIN is finished, the object has a Y rotation of 0. To get it back to our previous view, we enter the appropriate value for YROT again:

```
Ready 14 YROT
```

XROT and ZROT can, of course, be manipulated in identical ways.

Scaling

The image can be changed in width or height with the words SCALX and SCALY. Both of these words remove a number from the stack to select the given X or Y scale. The valid range is from -31 to +31. Numbers outside of this range will be "folded back" into the range. When the 3-D object array is initialized with ONJERASE, SCALX and SCALY are set to 16. Try these examples with "XYZ":

```
Ready 25 SCALX

Ready 8 SCALY

Ready 4 SCALX
```

Setting a scale of zero causes the object to have no "thickness" at all:

```
Ready 0 SCALX
```

Negative scale numbers reverse the image:

```
Ready -8 SCALX
```


Note: This reverse scaling is useful in unexpected ways. For example, if you are creating the image of a bird, you only need one wing image. The other wing is simply the first with one negative scale number to reverse the image.

Here's a programming example of scaling:

```
Ready : SQUASH 12 -12 DO I SCALX LOOP ;

Ready SQUASH
```

Since for most graphics applications you will want to change both the X and Y scale to change the total size of the object, the GraFORTH word SCALE is provided. SCALE has the same form as SCALX and SCALY. It simply sets both SCALX and SCALY to the same value:

```
Ready 5 SCALE

Ready 12 SCALE
```

Three-Dimensional Perspective

There is a fourth scaling word in GraFORTH, SCALZ. SCALZ doesn't change the size of the object in the same way that the other scaling words do; instead it changes the perspective of the object. Entries for SCALZ are also in the range -31 to 31. The default value for SCALZ is zero, which doesn't provide perspective views. (The front of a cube, for example, will be the same size as the back.) If you enter a nonzero number for SCALZ, perspective will be provided. If the entry is positive, the front of the object will be larger than the back. If the entry is negative, "reverse perspective" occurs, a most unusual phenomenon! You may wish to try the following examples:

```
Ready  20 SCALZ YSPIN

Ready -10 SCALZ YSPIN

Ready   0 SCALZ YSPIN
```

Note: When SCALZ is nonzero, images take about 20% longer to draw in exchange for the perspective features.

Also, SCALZ uses a fast algorithm that closely approximates true perspective. However, if you are displaying an image that has ends of lines meeting at the middle of a line, and you are using large amounts of perspective, the image may begin to distort. If this happens, break the image up into a series of shorter lines, so that all endpoints meet other endpoints, rather than meeting a line itself.

Position

Three-dimensional images can also be placed anywhere on the screen with the words XPOS and YPOS. XPOS and YPOS remove a number from the stack to determine the X or Y position on the screen of the center of the 3-D cube. Especially if the scale is large, to avoid screen wrap-around, ample room must be left on either side for the edges of the images. The valid entries for XPOS are 0 to 255; valid entries to YPOS are 0 to 191. The default values are 128 for XPOS and 96 for YPOS, which is the center of the screen.

To move the image around, let's first make it a bit smaller, to avoid wrap-around, then try a few different positions on the screen:

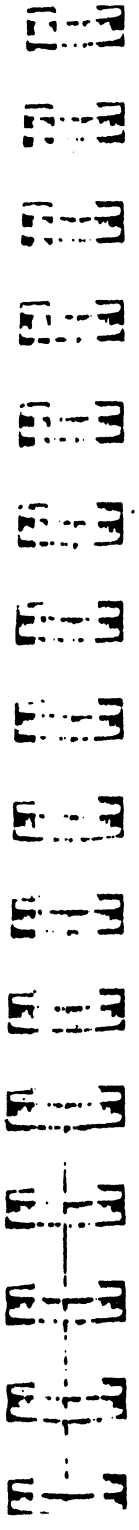
```
Ready 5 SCALE

Ready 50 XPOS

Ready  40 YPOS

Ready 200 XPOS
```

We can cause the feared wrap-around by placing the object close to one of the edges:

```
Ready 5 YPOS
```

Now let's move the image back to a more reasonable position:

```
Ready 96 YPOS
```

Translation

Translation occurs when the object is moved, not on the flat video screen, but within its own 3-dimensional space. In GraFORTH, objects can be translated along the X, Y, or Z axis with the words XTRAN, YTRAN, and ZTRAN. When using translation, you must keep the image inside the confines of its "cube of space". If you do not, then "3-D wrap-around" will occur, because GraFORTH cannot represent points outside of its cube of 3-D space.

Our current image, "XYZ" already reaches to the edges of its space on all three axes. We can translate it, but wrap-around will occur immediately:

```
Ready 5 XTRAN
```

For some examples of translation, let's first load another 3-D image, one that doesn't fill its space. We'll load and set up the image "HOUSE":

```
Ready ERASE

Ready CR 132 PUTC PRINT " BLOAD HOUSE,A3000 " CR

Ready 1 OBJECT  3000 ORJADR
```

The image of a house should appear. Let's get a better view:

```
Ready 20 XROT

Ready 10 YROT

Ready  8 SCALZ

Ready 10 SCALE
```

Now the house can be translated. It can be moved about a bit before causing wrap-around. (In the next section, you'll see how to determine the true size of an object from the IMAGEDITOR.)

```
Ready -50 ZTRAN

Ready  50 ZTRAN

Ready -25 XTRAN
```

Just for fun, try using YSPIN with the house, now that it has been translated away from the center of its space:

```
Ready YSPIN
```

Object Color

You noticed that each of the three arrows in "XYZ" was a different color. Images can be created with or without colors specified. If no color is specified, then the object's color can be determined when it is drawn later, using OBJCOLOR. OBJCOLOR removes a number from the stack to select the color of the current object. The usual GraFORTH color numbers are used.

The house does not have a set color, so we can set its color with OBJCOLOR:

```
Ready 1 OBJCOLOR

Ready 5 OBJCOLOR
```

Note that 3-D graphics, like two-dimensional and character graphics, can be done in either INVERSE or NORMAL, and either ORMODE or EXMODE, producing a wide variety of graphics effects. We encourage you to try some 3-D graphics commands with various combinations of display modes.

At the end of this chapter is a discussion of the program PLAY, which enables you to set all of these parameters (except for OBJCOLOR) into motion. PLAY is very useful in getting an intuitive feel for exactly what each of these parameters does.

The Image Editor

On the GraFORTH system disk is a file called IMAGEDITOR, which contains a program enabling you to create your own 3-D images. To use the image editor, first delete any new words on the word library to make room, then type:

```
Ready ABORT

Ready READ " IMAGEDITOR "
```


(NOTE: The image editor is a fairly large program. On non-language card systems, loading the image editor will move the top of the word library into the same memory used by the text editor program. If the editor is loaded into memory, it will overwrite the top of the word library, forcing you to reach for the power switch, as the GraFORTH system will become inoperable. After using the image editor, remember to FORGET the program before using the text editor.)

Now run the program:

```
Ready RUN
```

You will see a list of commands to the right and a prompt: "Enter command:". The image editor works with one 3-D image at a time.

Address and Image Selection

As in the character editor, you must select a work area address (or use the default address). To select an address, press "A" for "Address". You will see the prompt:

```
Enter File Address :
```

followed by the number "2816". (You should be getting pretty familiar with that number!) If you want to use another area of memory, enter that address. For this example, just hit <return>, and the address 2816 will be selected.

If you are doing these examples sequentially, the image "XYZ" will still be in memory at 2816. If you've turned the Apple off since that time, you will need to load it again. Type "G" for "Get" and enter the filename "XYZ". The file will be loaded into memory.

Getting a Good View

If the image was already in memory, it won't appear until you rotate it or move it on the screen. Images can be rotated, positioned, and scaled from the image editor.

To rotate the image, type "R". You will see:

```
Rotate [ X (num) to Z (num) ] :
```

For this command enter the letter of the axis you want to rotate around followed by the angle you want to rotate. For this example, type "Y16". The image will rotate around the Y-axis. Type "R" again and enter "X16". Now you can see the arrows well.

To scale the object, type "S". You will see the prompt:

Scale [(num), or X,Y,Z (num)] :

To scale X and Y simultaneously, simply enter a number. To scale one of the coordinates, type X, Y, or Z, and then the scale number. Since we're keeping the image in the corner of the screen, it's best to keep the scale small. The scale is initially set to 8.

To change the position of the object, type "P". You will see:

Position [X (num) or Y (num)] :

Enter an X or a Y followed by the desired screen position. The image has an initial screen position of X=64 and Y=48.

You can choose a color for the image, if the color is not already set in the image file. Press "C" for "Color" and enter the desired color number. You can also choose between EXMODE and ORMODE views. Press "M" for "Mode", then enter "X" for EXMODE or "O" for "ORMODE".

Image File Entries

Now type "L" for "List" to see the numbers that make up the image. You can press <return> to see all of the entries or press ConTRoL-C to stop. Remember that, as explained above, GraFORTH uses Cartesian coordinates, a system of three numbers for each defined point.

Each entry in the IMAGEDITOR listing has the following information:

1. Whether the point is to be (M) moved to without drawing, or (D) drawn to from the previous line ending. (This means that each image file must begin with (M), not (D), since there are no previous lines at that time.)

2. What color should be used for the line. The color number (if present) is directly under the letter "C" in the heading. (If it is desired to use the word "OBJCOLOR" to specify object color, then don't make any color entries within the image file.)

3. The X, Y, and Z coordinates of the point (each coordinate lies within the range -128 to 127).

4. The address of the entry. Each entry occupies four bytes.

The last six lines of the image file can also be seen by pressing "E" for "Enter". We will use the "Enter" command in a moment to create our own 3-D shape. For now, press <return> to leave the "Enter" mode.

While using the image editor, you may want more screen space for text and less for image drawing, or vice versa. To accomplish this you can use "W" to move the text window up or down, position the image using "P", and scale the image using "S". The "List" and "Enter" commands will use as many lines as the text window allows.

Sometimes, while adjusting the image position, the image will "wrap around" on the graphics screen. If you want to clean up the screen, type "W" and reenter 14 or some other window top value. "W" clears the screen when it sets a new window.

Creating New Images

Now we will create our own image, a cube. First, we need to erase "XYZ". Press "Z", and you will see:

Erase File (Y/N) :

Type a "Y" to erase the file. The image won't disappear right away. (If the presence of the old image disturbs you, press "W" and enter 14 to cause the "Window" command to erase the screen.)

So that we will be able to see all sides of our object as it is created, enter a Z scale of 8 for perspective (press "S", then "Z8"). Now press "E" again. Notice that no file entries are listed, since we have erased them. You will see a prompt:

(M)ove, (D)raw, (-) Delete, (CR) Quit :

Since the first entry must be a move, type "M". You will be prompted for a color. Let's not use a color, so that later we can select its color with OBJCOLOR. Just press <return>.

You will then be prompted for X, Y, and Z values in turn. We're going to start with the point at the lower left front corner of the cube. X at the left is -127, so enter -127 and press <return>. Y at the bottom is 127. Enter 127 and press <return>. Z at the front is 127, so enter that and press <return>.

You still won't see anything drawn, because we have only defined a single point, and points aren't plotted in GraFORTH 3-D graphics, only lines. Now let's draw our first line. Type "D" this time instead of "M". Now enter an X value of 127 (remember the last entry was -127). We want the other two values to stay the same. In this "Enter" mode, to keep a previous value, just press <return>. The last value will be repeated. Press <return> for both Y and Z. Now a line will appear from left to right (from X = -127 to X = 127).

Now repeat the entry procedure, pressing "D" each time and changing only one number per entry, pressing <return> for the others:

Z to -127
X to -127
and Z to 127 again.

These entries will draw a square at the bottom of the image space. (If the view isn't very good, press <return> to leave "Enter" mode, change the rotation or the scaling, then press "E" to return to "Enter" mode.)

Note: If at any time you make an incorrect entry, just finish the entry, then press "-". "-" deletes the last entry in the file.

Now if we change Y to -127 and repeat the entire procedure, we will have most of the cube.

At this point three edges are still missing. Can you figure out how to draw the missing edges?

The solution is to (M)ove to each of the following locations, and (D)raw a vertical line (using Y) from bottom to top:

1. (M) X = 127, Y = 127, Z = 127
2. (D) X (same), Y = -127, Z (same)
3. (M) X (same), Y = 127, Z = -127
4. (D) X (same), Y = -127, Z (same)
5. (M) X = -127, Y = 127, Z (same)
6. (D) X (same), Y = -127, Z (same)

Saving the Image File

Now we can save our cube. Press <return> with no entry to leave the "Enter" mode, then press "K" for "Keep". You will be prompted:

Enter File Name to Keep :

Enter a file name here. The GraFORTH system diskette already contains a file named "CUBE". (It contains a cube identical to the one we just made here.) If you're using another disk, you can use the filename "CUBE" or another filename.

Three-Dimensional Display Methods

From within a program, the word DRAW is usually used instead of AUTODRAW to draw 3-D images. This way, several parameters can be changed at once before the next image is drawn. When AUTODRAW is off, executing DRAW causes the images to be drawn.

Aside from the mathematical methods (described in Appendix B), GraFORTH has a rather complex display method for 3-D images. In general, when a DRAW command is issued, the following events occur:

1. The drawing routines are directed at the graphics screen that is not currently being displayed, so that the drawing won't be seen.

2. The previous image on the invisible screen is "undrawn", using information stored when it was drawn.

3. The new image is drawn.

4. The display is switched to the freshly drawn screen.

This method guarantees high-quality animation images, since the entire process of drawing is concealed from the viewer.

You may wish to note that character graphics, discussed in the last chapter, also draws to both screens, so that character and 3-D graphics can be freely intermixed.

Redrawing Without Change

For maximum speed, an object is only redrawn by DRAW if a new command is issued to it. So in a program with several objects, only those that have been referenced since the last DRAW will be redrawn. Example:

```
0 OBJECT  16 XROT
3 OBJECT  24 YROT
DRAW
```

Only objects 0 and 3 will be redrawn when DRAW is executed.

If an object has been changed and then drawn, the images of the object on the two graphics screens will not be the same. If other objects are then repeatedly changed and drawn, causing GraFORTH to switch graphics screens, then the two unlike images of the object will be alternated, causing a back-and-forth type of residual motion.

Therefore, if several objects are being drawn independently, they should be referenced (using the word OBJECT), if not changed, to cause the image to be redrawn. This way, the images on both graphics screens will always be updated. For example,

```
1 OBJECT
```

causes a redraw of object 1 at the next draw command.

Erasing Individual Objects

The GraFORTH word OFF is used to "undraw" an object but not redraw it. Most objects stay on the screen after the last image entry to their tables. OFF selectively erases objects that are no longer needed. Subsequent commands to an object will redraw it. Here is an example of OFF:

```
Ready 3 OBJECT OFF
```

Overlapping Objects and UNDRAW

In a case where there are several overlapping objects, or objects are drawn over text, it is best to use "EXMODE", since this causes drawing and undrawing to occur without destroying the screen's original contents. Alternatively, if all the objects are in continuous motion, it may be desirable to use the word UNDRAW.

UNDRAW simply erases a block of character spaces specified by BLKSIZE, just as UNBLK does. However, UNDRAW also causes the next DRAW command to not do an automatic line "undraw" before drawing the next image. This way, you can use UNDRAW to erase the 3-D images yourself. Using UNDRAW is frequently faster than the automatic line undraw that is carried out by DRAW.

For example, let us say we have an image in the center of the screen (at X = 128, Y = 96) that extends 20 plotting points in radius around this point. Remember that numbers entered to BLKSIZE refer to characters, not points. Text characters of size 0 are 7 points wide and 8 points high. So an entry to BLKSIZE of 6 by 5 will cover an area 42 by 40 points, large enough for our sample image. Remember that UNDRAW, like UNBLK, is controlled by VTAB and HTAB. Let's set the blocksize, then position and execute an UNDRAW before the next DRAW:

```
Ready 6  5 BLKSIZE

Ready 18 VTAB  17 HTAB UNDRAW DRAW
```

Remember also that UNDRAW, like PUTRLK and UNRLK, doesn't advance HTAB across the screen as for printing. Once positioned, UNDRAW can be used repeatedly over the same area.

Other Effects

If you wish to prevent undrawing of the images (for special effects), simply use UNDRAW, but place the undraw block away from the image. For speed, select a blocksize of 1 by 1 in this case.

It is also possible to prevent screen sequencing altogether, using SEQUENCE, so that the process of drawing may be observed. SEQUENCE removes a number from the stack. If this number is a 0, screen sequencing is turned off. If the number is 1, screen sequencing is turned back on. This example will stop screen sequencing:

```
Ready 0 SEQUENCE
```

Usually used with "0 SEQUENCE", the word "SCREEN" selects which graphics screen to display. The screens are numbered 0 and 1. This example displays screen number 1:

```
Ready 1 SCREEN
```

PROFILE

There is another program on the GraFORTH system disk used for creating 3-D images, called PROFILE. PROFILE acts as a sort of graphics "lathe", creating images that are cylindrical in nature from a set of points defining the profile of the image. The file "CHAL" on disk contains the image of a chalice, and is an example of the kinds of images that can be created with PROFILE.

To run PROFILE, first make sure that there is room on the word library by FORGETting any extra words, then type:

```
Ready READ " PROFILE "

Ready RUN
```


Setting Parameters

You will see the PROFILE heading and some instructions. We're going to use PROFILE in this example to create a simple cone. The first question asked is:

```
Enter number of polygon sides :
```

This determines how smooth the cones circumference will be. For a perfect circle, you would ideally want to enter an infinite number of sides. Unfortunately, your Apple does not contain an infinite amount of memory! For this example, enter a 20.

The next prompt reads:

```
Enter Object File Address :
```

with a good ol' 2816 already selected for you. Images created with PROFILE can easily use a lot of memory. Usually you will want to use the area of memory beginning at 2816 or the space above the word library. (To find this address, print the value of PRGTOP after loading PROFILE, and add about 50 or 100 to this address for extra space.) For this example, just press <return> to keep the address 2816.

Entering Data from the Keyboard

Now you will see:

```
Data from [K]eyboard or [D]isk ?
```

You can either enter the profile coordinates directly from the keyboard or use a text file that contains the coordinates. Here we will enter the coordinates directly. Press "K" for "Keyboard". You will see:

```
Enter X,Y pair (end = "E") :
```

This is where you actually enter the coordinates. The Y coordinate is the vertical position in the profile. The valid range is -128 to 127. The X coordinate can actually be considered a radius, since it determines the distance from the edge to the center of the object. Its valid range is also -128 to 127, but negative entries are identical to positive ones, so only numbers from 0 to 127 need be used.

We're going to start our cone as a single point, and work down. The top of the cone is at Y = -128, and the radius (X) is zero. As we move down with increasing Y values, we'll also steadily increase the radius. Make the following entries:

```
Enter X,Y pair (end = "E") : 0,-128
Enter X,Y pair (end = "E") : 32,-64
Enter X,Y pair (end = "E") : 64,0
Enter X,Y pair (end = "E") : 96,64
Enter X,Y pair (end = "E") : 127,127
Enter X,Y pair (end = "E") : E
```

The last entry must be "E". For a few seconds, the phrase:

Generating image file (824 bytes) . . .

will appear on the screen as PROFILE computes the points that make up the cone, then the screen will be erased and the cone will appear. Notice that the cone has 20 vertical lines around its circumference. This is because we selected 20 polygonal sides. There are 4 circles around the cone and a point at the top. These are because we made 5 profile entries. At the bottom of the screen will be the message:

Enter object file name :

This is so you can save the 3-D object to disk. If you want to save the cone to disk, enter a filename and press <return>. If you don't want to save the image, just press <return> and the program will end.

Entering Data from Disk

As discussed earlier, PROFILE can also read a list of coordinates from a disk file. The textfile "BIGCHAL" contains a list of coordinates that describes the profile of a chalice. You may wish to see this list at some point. When PROFILE is no longer in memory, you can enter the text editor, get the file BIGCHAL, and list it. You will see a list of numbers similar to the one we entered to make the cone, but longer. Note that the last

entry in the file is "E", marking the end of the list. For now though, let's run PROFILE again, this time using the textfile BIGCHAL instead of keyboard entries. RUN the program, select 8 polygon sides, the address 2816, then "D" to read data from disk. You will then be prompted:

Enter Data File Name :

Enter the name "BIGCHAL". The disk will whir for a bit, then the message:

Generating image file (2724 bytes) ...

will appear. After a pause, the chalice will appear on the screen. As before, you can either save the 3-D image to disk, or press <return> to exit.

Memory Considerations

Because PROFILE can generate very large image files rapidly, image size checking has been added to help prevent overwriting important parts of memory.

Usually you will use one of two areas of memory for the 3-D image file when using PROFILE: either the free space from locations 2816 to 5887, or the space above the top of the word library. If you select an address between 2816 and 5887, PROFILE will prevent the image from extending beyond location 5887.

If you select an address greater then 5887, then PROFILE assumes the image is above the word library. It then checks for the presence of a language card. If you are using a language card, PROFILE will allow images to extend to location -16385, immediately below the Apple][I/O area. If you do not have a language card, PROFILE prevents the image from extending beyond location -26113, immediately below DOS.

If the image is too large to fit in the provided space, the image will not be created or drawn, and the following message will appear:

Not enough room here.
(Requires nnnn bytes.)

with nnnn being the actual number of bytes the image requires.

Notice that if the starting address you select is in a "safe" area of memory, then PROFILE will prevent the image from clobbering important information. However, if you select an address in the middle of something important, you'll find yourself having to reboot the system from scratch....

PLAYing Around

The program PLAY was briefly introduced in Chapter 1. PLAY was designed for you to "play" with a 3-D image, manipulating its rotation, scale, translation, and position parameters. Any or all of these parameters can be set into motion, giving you a rapid intuitive "feel" for what each of the parameters does. And PLAY is a lot of fun!

Note that PLAY, like IMAGEDITOR, uses the same memory as does the text editor on non-language card systems. Be sure to forget any extra words in the word library (PLAY is rather a large program), then type:

```
Ready READ " PLAY "

Ready RUN
```

The instructions are fairly self-explanatory. Once the image is loaded and you begin "playing", you can select a parameter with one of the number keys. To set the parameter in motion, press one of the arrow keys. The right arrow increases the parameter value; the left arrow decreases it. By pressing several number keys and arrow keys alternately, you can set a number of parameters in motion at once.

If any one parameter gets out of hand, you can press "F" to "freeze" its motion, leaving it at the current value. You can also press "D", to bring it back to its "Default" value.

If you want to pause everything, just press CONTROL-S. The display will pause, and a flashing cursor will appear in the upper-left corner. Just press any key to resume. If you want to bring everything to a complete halt, press ESC. All motion will stop and all parameters will be set back to their default values. Finally, typing "?" will display the instruction screen again, and "Q" will quit the program.

Let's answer the start-up questions and get things moving:

The first prompt you will see is:

```
Image in [M]emory or on [D]isk?
```

If you already have an image in memory, press "M". If you want to load an image from disk now, press "D". For this example, press "D". Next is the now-famous address question:

```
Enter image address :
```

again with the number 2816 waiting for you. If you want to use the address 2816, just press <return>; otherwise enter the address you want. Press <return> for this example. If you selected to load an image from disk a moment ago, you will then see:

```
Enter image filename :
```

Type the name of the file you want to load. Let's load the file "MOUSE". Lastly:

```
Press Return to begin...
```

The screen will be erased and the image will appear. Along the right side are the values for each of the parameters. When you press a number key, the selected parameter will also be displayed on the bottom line with its current value and increment. Pressing the arrow keys will change the increment and set the object in motion.

You'll also see a question mark in the lower right corner. This is just to remind you that the instructions can be displayed at any time by typing "?".

With PLAY, it's very easy to get some of the parameters out of bounds, causing screen or "space" wrap-around. It doesn't hurt anything, and it can sometimes produce rather amusing effects!

Conclusion

We've now looked at all three kinds of graphics: two-dimensional graphics, character graphics, and three-dimensional graphics. With the information presented in these chapters, you can incorporate a wide variety of animated color graphics effects into your own programs, then use SAVEPRG to produce a system that boots and runs them automatically!

The next chapter explains how you can create music and sound effects with GraFORTH. (We'll also mention another program you may be interested in...) So without any further delay, on to chapter 9!

CHAPTER NINE: MUSIC WITH GRAFORTH

Chapter Table of Contents:

Introduction

GraFORTH has a sophisticated music synthesizer that plays through the Apple][built-in speaker. Notes may be played in nine distinct voices (not simultaneously). These features allow you to incorporate music or sound effects into your applications or game programs.

The two GraFORTH words that control the synthesizer are VOICE and NOTE.

VOICE

The GraFORTH word VOICE selects one of 9 voices in which to play notes. VOICE removes a number from the stack, and uses it to select a given voice. Here are the VOICE numbers and their meanings:

Number	Voice
-6 to -1	Selects a constant 'duty cycle' for the note, producing a note that is constant in volume. -1 = 50% duty cycle, -2 = 25% duty cycle, -3 = 12.5% duty cycle, etc. Smaller duty cycles decrease volume and increase the amount of high-frequency energy in the note.
0	Note begins at 50% duty cycle, then decreases to 0%. The note seems to die away.
1	The note begins at 0%, increases to 50%, then decreases again.
2	The note begins at 0%, then increases to 50%. The note seems to increase in volume.

NOTE

The GraFORTH word NOTE actually causes a note to be played. NOTE removes two numbers from the stack to select pitch and duration, then plays the note. The form for NOTE is:

<pitch> <duration> NOTE

The valid numbers for pitch and duration are in the range 2 to 255. Larger numbers for duration produce longer notes. Larger numbers for pitch produce lower pitched notes.

Let's play a couple of notes. The voice used if one has not been selected is voice 0. This example plays an "A" two octaves below middle A:

```
Ready 124 255 NOTE
```

Let's try a different note:

```
Ready 62 128 NOTE
```

This plays a note an octave higher for half as long. Now let's change the voice and play the same note:

```
Ready -1 VOICE

Ready 62 128 NOTE
```

Notice the change in tone quality. Experiment with the different voices to hear their differences.

Determining Duration and Pitch

The duration of a note is directly related to the size of the duration number. 255 can be considered a whole note, 128 a half note, 64 a quarter note, and so forth. Of course, if you want to play notes at a faster tempo, simply use smaller numbers.

Here is a table relating notes to the pitch numbers which produce them:

Note	Octave 1	Octave 2	Octave 3	Octave 4
A	249	124	62	31
A#	234	117	58	29
B	221	110	55	27
C	209	104	52	26
C#	197	98	49	24
D	186	93	46	23
D#	175	87	43	21
E	166	83	41	20
F	156	78	39	19
F#	147	73	36	18
G	139	69	34	17
G#	131	65	32	16

Useful Music Words

If you don't want to look up the pitches for each note, you can use the following program to generate the table and store it in a string array called "PITCH". Each element of PITCH, instead of containing a character, contains the pitch value for a note.

```
50 STRING PITCH

: COMPUTE.NOTES
  24870
  48 0 DO
     DUP 100 / I PITCH POKE
     DUP 18 / -
     DUP 1655 / -
  LOOP DROP ;

Ready COMPUTE.NOTES
```

Running COMPUTE.NOTES generates the table in PITCH. Now the pitch values for the 48 notes (numbered 0 through 47) can be found by reading the value from the proper element of PITCH. For example, the pitch value for the note 3 in the table (a "C" from the first octave) can be found in position number 3 in PITCH:


```
Ready 3 PITCH PEEK .
209
```

To play this note as a half note, you can enter:

```
Ready 3 PITCH PEEK 128 NOTE
```

You can also define a short word that retrieves the pitch value for you:

```
Ready : GETPITCH  PITCH PEEK . ;

Ready 3 GETPITCH
209
```

This word can be used with NOTE:

```
Ready 3 GETPITCH 128 NOTE
```

Since the notes are now numbered from 0 to 47, we can play all of the notes in the scale by using a loop:

```
Ready 48 0 DO I GETPITCH 32 NOTE LOOP
```

With a little patience, we can put together a song! The following word definition plays the first phrase from the "Happy Birthday" song:

```
: HAPPY.B
  12 GETPITCH 50  NOTE
  12 GETPITCH 50  NOTE
  14 GETPITCH 100 NOTE
  12 GETPITCH 100 NOTE
  17 GETPITCH 100 NOTE
  16 GETPITCH 200 NOTE ;
```

For longer tunes, repeating the words GETPITCH and NOTE will waste a lot of space. We wanted to show here how simply the tunes can be constructed. A much more efficient method is to store the numbers in memory or on the stack, and read them and play the notes from a loop.

Postscripts

Note: The quality of the synthesizer is higher than can be demonstrated with the Apple][built-in speaker. The use of a large external speaker is recommended for serious music work. See the Apple][Reference Manual or your local dealer for connection information.

For two-part music applications, the Electric Duet, also written by Paul Lutus, is available from Insoft. The Electric Duet plays 2 simultaneous notes through either the Apple speaker or an external amplifier, and can be used to play music directly from your GraFORTH programs. It contains a full feature music editor with the ability to transpose both note pitch and duration. Music can be directed to either the internal speaker or the Apple][tape output jack. The suggested price of the Electric Duet is only $29.95. For more information, contact Insoft or your local Apple dealer.

CHAPTER TEN: FINAL WRAP

We've made it! You have now been introduced to the GraFORTH system, from language features to complex graphics. From here on out, you will probably be using this manual more as a reference guide than as a tutorial; therefore, we suggest you get acquainted with the appendices. You will find the Word Library listings invaluable, and the Index very helpful for finding those definitions you've forgotten. The technical data section covers very useful information we suggest you at least browse through, and the GraFORTH diskette file listing and ASCII code tables are excellent references when you need them.

Please note that if you are using or intend to use GraFORTH to develop software for re-sale, we would like to talk with you. Insoft represents fine software (such as this!) for Apple, IBM, Atari, NEC and other popular microcomputers. Our royalty rates are among the best in the industry, and our support team is second to none. Let us show you why using our team of professionals makes good sense!

If you decide to market software on your own, please call us for information on a license agreement to use GraFORTH. There is no fee for this license, however, we do have a few restrictions on how it is marketed (We'll show you how to lock GraFORTH so that only your program can be run.) Either way, please contact:

Michael Brown
Insoft
10175 SW Barbur Blvd. Suite 202B
Portland, Oregon, 97219
(503) 244-4181

You now have a graphics system that is quite nearly limited only by your imagination! We hope you enjoy learning and using GraFORTH as much as we have enjoyed the opportunity to bring it to you!

APPENDIX A:
WORD LIBRARY LISTING

The following is a list of the words in the GraFORTH word library. The list includes the word name, a "before and after" stack picture, the page number in the text where the word is first introduced, and a brief description of what the word does.

The stack picture shown represents relevant numbers on the top of the stack as letters. The top of the stack is to the right, as indicated by a dash. Three dashes represent an empty stack. How words use the stack can usually be inferred simply from the stack picture.

The word descriptions here are not meant to be comprehensive. For more information on each word, we suggest you refer back to the text, using the page numbers provided.

GraFORTH WORD LIBRARY LISTING

Word Name	Before	After	Page
"	- - -	- - -	3-13

A set of quotes surrounding text causes the text to be compiled into the program. Used with PRINT, ASSIGN, and READ.

$LIST	- - -	- - -	5-30

Lists words in word library with hexadecimal addresses.

'	- - -	a -	5-30

a = address of the word that follows ', and prevents that word's execution.

(	- - -	- - -	4-14

Indicates the beginning of a program comment, to be passed over by the GraFORTH compiler.

Word Name	Before	After	Page
*	m n -	p -	3-10
p = m * n (multiplication)			
+	m n -	p -	3-6
p = m + n (addition)			
+LOOP	n -	- - -	3-20
Marks the end of a loop structure, using n as a loop value increment.			
,	- - -	- - -	5-32
Compiles a single byte within word definitions.			
-	m n -	p -	3-10
p = m - n (subtraction)			
->	(not applicable)		5-8
Causes the next variable reference to store the top stack value into the variable, rather than placing the variable value on the stack.			
.	n -	- - -	3-6
Prints n.			
/	m n -	p -	3-10
p = m / n (division)			
:	- - -	- - -	3-14
Marks the beginning of an executable word definition.			
;	- - -	- - -	3-14
Marks the end of a word definition.			
<	n m -	p -	3-23
p = 1 if n < m, otherwise p = 0.			
<=	n m -	p -	3-23
p = 1 if n <= m, otherwise p = 0.			

Word Name	Before	After	Page
<>	n m -	p -	3-23
p = 1 if n <> m, otherwise p = 0.			
=	n m -	p -	3-23
p = 1 if n = m, otherwise p = 0.			
>	n m -	p -	3-23
p = 1 if n > m, otherwise p = 0.			
>=	n m -	p -	3-23
p = 1 if n >= m, otherwise p = 0.			
ABORT	- - -	- - -	7-3
Restarts GraFORTH from scratch. The screen is erased, character size of 0, color of 3, all stack pointers initialized to 0.			
ABS	n -	m -	3-10
m = absolute numeric value of n.			
AND	n m -	p -	3-23
p = 1 if both n and m are nonzero, otherwise p = 0.			
AREG	(variable)		5-31
Value of AREG is placed in processor A register before a CALL. After CALL, contents of A register are loaded back into AREG.			
ASSIGN	a -	- - -	5-12
Places following quoted text into memory starting at address a.			
AUTODRAW	n -	- - -	8-3
If n is nonzero, 3-D objects will automatically be drawn after every graphic command. If n is zero, this feature is turned off.			
AUTORUN	n -	- - -	5-26
If n is nonzero, the top word library word will automatically execute at every return to the system. If n is zero, this feature is turned off.			

Word Name	Before	After	Page
BASE	(variable)		5-22
Value determines what base numbers are accepted and displayed in.			
BEGIN	- - -	- - -	3-29
Provides a program return point for the words REPEAT and UNTIL.			
BELL	- - -	- - -	3-3
Beeps the Apple speaker.			
BINARY	- - -	- - -	5-22
Sets number input and output to base two.			
BLKSIZE	h v -	- - -	7-12
Selects a blocksize of h characters horizontally by v vertically for use by PUTBLK, UNBLK, and UNDRAW.			
BYE	- - -	- - -	5-32
Exits GraFORTH to Apple monitor.			
CALL	a -	- - -	5-31
Loads processor registers from AREG, XREG, YREG, AND PREG, calls machine language routine at address a, then stores register values.			
CASE:	n -	- - -	3-32
Selects and executes nth following word from list of words numbered starting from 0.			
CHRADR	a -	- - -	7-6
Selects a as address of current character set.			
CHRSET	(variable)		7-6
Value is address of default character set (2048).			
CHRSIZE	n -	- - -	7-3
Selects character size for subsequent character printing using PRINT, WRITELN, PUTC, and PUTBLK.			

Word Name	Before	After	Page
CHS	m -	n -	3-10
n = - m (change sign)			
CLEOL	- - -	- - -	5-4
Clears from the cursor position to the end of the current line.			
CLEOP	- - -	- - -	5-4
Clears from the cursor position to the end of the text window.			
CLOSE	- - -	- - -	5-24
Causes DOS to close any open files.			
CLRKEY	- - -	- - -	5-20
Clears the Apple][keyboard strobe so that a key can be read with GETKEY.			
COLOR	n -	- - -	6-6
Selects the color for line and large character drawing.			
CR	- - -	- - -	3-13
Prints a carriage return (ASCII value 141).			
DECIMAL	- - -	- - -	5-22
Sets number input and output to base ten.			
DO	m n -	- - -	3-19
Initializes a loop, using n for an initial value and m as an ending value.			
DRAW	- - -	- - -	A-15
Causes all 3-D objects referenced since the last DRAW to be drawn, using 3-D display methods.			
DROP	n -	- - -	3-7
Discards n from the stack.			
DUP	n -	n n -	3-7
Makes a copy of n on the stack.			

Word Name	Before	After	Page
EDIT	- - -	- - -	4-2
Loads from disk (if necessary) and runs the appropriate text editor.			
ELSE	- - -	- - -	3-27
Separates the two controlled areas in an IF - ELSE - THEN construct.			
EMPTY	x y -	- - -	6-8
Erases a rectangular area from the last plotted point to (x,y).			
ERASE	- - -	- - -	5-4
Erases both graphics screens.			
EXMODE	- - -	- - -	6-10
Causes plotted points to turn on corresponding screen locations that are off, and turn off locations that are on.			
FILL	x y -	- - -	6-8
Fills a rectangular area from the last plotted point to (x,y).			
FORGET	- - -	- - -	3-17
Truncates the GraFORTH library back to the word that follows FORGET.			
GETC	- - -	n -	5-20
Gets a single character from the keyboard, placing its ASCII value on the stack.			
GETKEY	- - -	n -	5-20
Reads the keyboard without waiting, returning an ASCII value. Values over 128 are valid. Should be followed by CLRKEY.			
GETNUM	a -	n -	5-14
Converts text string at address a into a number. Unsuccessful conversions return 0.			

Word Name	Before	After	Page
GPEEK	x y -	n -	6-12
Examines point at screen coordinates (x,y). n is nonzero if point is turned on, or 0 if point is turned off.			
GR	- - -	- - -	3-38
Reestablishes normal GraFORTH input and output, and sets the graphic display mode.			
HEX	- - -	- - -	5-22
Sets number input and output to base 16.			
HOME	- - -	- - -	5-4
Erases the screen inside the text window and sets HTAB and VTAB to the upper left corner of the window.			
HTAB	h -	- - -	5-3
Sets the column for subsequent printing.			
I	- - -	n -	3-19
Returns the current innermost loop value.			
IF	n -	- - -	3-25
If n is nonzero, words between IF and THEN (or IF and ELSE) are executed, otherwise execution continues after THEN (or between ELSE and THEN).			
INVERSE	- - -	- - -	6-9
Complements the color for all text and graphics displays (including black-on-white text).			
J	- - -	n -	3-20
Returns the loop value for the next outer loop.			
K	- - -	n -	3-21
Returns the loop value for the third outer loop.			
LINE	x y -	- - -	6-4
Draws a line from the last plotted point to (x,y).			

Word Name	Before	After	Page
LIST	- - -	- - -	3-3
Lists the words in the GraFORTH word library.			
LOOP	- - -	- - -	3-19
Marks the end of a loop structure, incrementing the loop value and looping back to the word after DO if the loop value is less than the ending value.			
MAX	m n -	p -	3-10
p = the greater of m or n.			
MEMRD	a -	- - -	4-13
Reads and compiles text in memory starting at address a.			
MIN	m n -	p -	3-10
p = the smaller of m or n.			
MOD	m n -	p -	3-10
p = remainder after dividing m by n.			
MOVMEM	a b n -	- - -	5-30
Moves a block of n bytes from address a to address b.			
NORMAL	- - -	- - -	6-9
Resets normal color (white-on-black text) display.			
NOTE	p d -	- - -	9-3
Sounds a note of pitch p and duration d in the current voice.			
OBJADR	a -	- - -	8-3
Selects a as address of currently selected 3-D object.			
OBJCOLOR	n -	- - -	8-10
Selects color of current 3-D object.			
OBJECT	n -	- - -	8-3
Selects which object subsequent 3-D commands will refer to.			

Word Name	Before	After	Page
OBJERASE	- - -	- - -	8-3
Initializes the 3-D image array. Should be used at the beginning of 3-D graphics programs.			
OFF	- - -	- - -	8-17
Causes the next DRAW command to undraw the 3-D object.			
OR	m n -	p -	3-23
p is bit-wise OR of m and n. (p is nonzero if either m or n is nonzero, otherwise p = 0.)			
ORMODE	- - -	- - -	6-10
Causes points to be plotted regardless of what screen locations are on or off.			
OVER	m n -	m n m -	3-7
Copies m to top of stack.			
PAD	- - -	a -	5-15
Returns the address (812) of a 120-byte string space.			
PEEK	a -	n -	5-6
Reads a single byte n from address a.			
PEEKW	a -	n -	5-6
Reads number n from address a.			
PICK	..m n -	..m p -	3-7
Copies the nth stack item to top of stack.			
PLOT	x y -	- - -	6-4
Plots a point at (x,y).			
POKE	n a -	- - -	5-6
Stores single byte n at address a.			
POKEW	n a -	- - -	5-5
Stores number n at address a.			

Word Name	Before	After	Page
POP	- - -	- - -	3-22
Discards top return stack value.			
POSN	x y -	- - -	6-6
Establishes a position for a "last plotted point" without plotting.			
PREG	(variable)		5-31
Value of PREG is stored in processor status register before a CALL. After CALL, value of status register is stored back into PREG.			
PRGTOP	- - -	a -	3-3
Returns the address of the top of the word library.			
PRINT	- - -	- - -	3-13
Prints following quoted text.			
PULL	- - -	n -	3-22
Moves top return stack value to data stack.			
PUSH	n -	- - -	3-22
Moves top data stack value to return stack.			
PUTBLK	n -	- - -	7-13
Draws a block of characters with present blocksize starting with character number n at the current cursor position.			
PUTC	n -	- - -	5-19
Prints character with ASCII value n at the current cursor position.			
READ	- - -	- - -	4-14
Reads and compiles text from file with following quoted filename.			
READLN	a -	- - -	5-12
Reads a line from keyboard into string starting at address a.			

Word Name	Before	After	Page
REPEAT	- - -	- - -	3-31
Marks the end of the BEGIN - WHILE - REPEAT construct, causing execution to jump back to words following BEGIN.			
RND	- - -	n -	3-10
n is a random number.			
RNDB	- - -	n -	3-10
n is a random number from 0 to 255.			
RUN	- - -	- - -	5-26
Executes the top word on the word library.			
SAVEPRG	- - -	- - -	5-27
Saves current system to disk.			
SCALE	n -	- - -	8-7
Sets the X and Y scales for the current 3-D object.			
SCALX	n -	- - -	8-6
Sets the X scale (width) for the current 3-D object.			
SCALY	n -	- - -	8-6
Sets the Y scale (height) for the current 3-D object.			
SCALZ	n -	- - -	8-6
Sets the Z scale (perspective) for the current 3-D object. Faster drawing occurs with a SCALZ of 0.			
SCREEN	n -	- - -	8-18
Selects display of the given graphics screen (0 or 1).			
SEQUENCE	n -	- - -	8-18
If n = 1, automatic screen sequencing for 3-D drawing is enabled. If n = 0, sequencing is enabled. (Default=1)			
SGN	m -	n -	3-10
n = 1 if m > 0, 0 if m = 0, -1 if m < 0.			

Word Name	Before	After	Page
SIN	n -	n -	3-10
n is a scaled sine value for m, in the range -128 to 127, repeating for every 128 numbers.			
SPCE	- - -	- - -	3-13
Prints a space (ASCII value 160).			
STACK	- - -	- - -	3-5
Toggles the stack display on or off.			
STRING	- - -	- - -	5-9
Declares a string array with following name, setting aside number of characters specified before STRING.			
SWAP	m n -	n m -	3-7
Swaps position of top two stack values.			
TEXT	- - -	- - -	3-38
Reestablishes normal GraFORTH input and output, and sets text display mode (no graphics).			
THEN	- - -	- - -	3-25
Marks the end of an IF - THEN construct, where execution continues from.			
UNBLK	- - -	- - -	7-14
Erases a block with present blocksize at the current cursor position.			
UNDRAW	- - -	- - -	8-17
Erases a block and prevents the next DRAW from performing an automatic line undraw.			
UNLINE	x y -	- - -	6-8
Erases a line from the last plotted point to (x,y).			
UNPLOT	x y -	- - -	6-8
Erases a point at (x,y).			
UNTIL	n -	- - -	3-29
If n = 0, execution jump back to words that follow BEGIN.			

Word Name	Before	After	Page
VALID	- - -	n - -	5-14
n is nonzero if last GETNUM produced a valid number, otherwise n = 0.			
VARIABLE	- - -	- - -	5-7
Declares a variable with following name. Any preceding number is used as the variable's initial value.			
VOICE	n -	- - -	9-2
Sets the voice for subsequent NOTE commands. Valid numbers are -6 to 2.			
VTAB	n -	- - -	5-3
Sets the row for subsequent printing.			
WHILE	n -	- - -	3-31
If n is nonzero, execution continues after WHILE, otherwise execution jumps to words after REPEAT.			
WINDOW	L w t b -	- - -	5-3
Sets a text window with left margin L, width w, top margin t, and bottom margin b.			
WRITELN	a -	- - -	5-12
Writes text to screen from string at address a.			
XPOS	n -	- - -	A-8
Sets X-position of current 3-D object to n.			
XREG	(variable)		5-31
Value of XREG is placed into processor X register before a CALL. After CALL, value of X register is stored back into XREG.			
XROT	n -	- - -	A-5
Sets rotation of current 3-D object around X-axis to n.			
XTRAN	n -	- - -	A-9
Translates current 3-D object along X-axis by n.			

Word Name	Before	After	Page
YPOS	n -	- - -	8-8
Sets Y-position of current 3-D object to n.			
YREG	(variable)		5-31
Value of YREG is placed into processor Y register before a CALL. After CALL, value of Y register is stored back into YREG.			
YROT	n -	- - -	8-5
Sets rotation of current 3-D object around Y-axis by n.			
YTRAN	n -	- - -	8-9
Translates current 3-D object along Y-axis by n.			
ZROT	n -	- - -	8-5
Sets rotation of current 3-D object around Z-axis by n.			
ZTRAN	n -	- - -	8-9
Translates current 3-D object along Z-axis by by n.			

APPENDIX A: WORD LIBRARY BY SUBJECT GROUP

Numeric Operator Words

CHS	ABS	SGN	RND	RNDM
MIN	MAX	POKEW	POKE	<>
=	>	<	>=	<=
OR	AND	PEEKW	PEEK	SWAP
DROP	POP	I	J	K
PULL	PUSH	DUP	OVER	PICK
MOD	/	*	+	-
SIN	BASE	DECIMAL	BINARY	HEX
MOVMEM	VALID	GETNUM		

Program Branching or Control Words

+LOOP	LOOP	DO	REPEAT	WHILE
UNTIL	BEGIN	IF	THEN	ELSE
BYE	STACK	FORGET	VARIABLE	RUN
AUTORUN	ABORT	READ	MEMRD	;
:	CASE:	(	CLOSE	EDIT
PRGTOP	SAVEPRG	->		

Input/Output Operator Words

HOME	CLEOP	CLEOL	GETC	GETKEY
CLRKEY	PUTC	.	$LIST	LIST

Text Display Function Words

VTAB	HTAB	CHRADR	CHRSET	CR
SPCE	TEXT	WINDOW	PRINT	ASSIGN
"	STRING	PAD	READLN	WRITELN

General Graphics Words

GR GPEEK ORMODE EXMODE ERASE
COLOR INVERSE NORMAL

Two-Dimensional Graphics Words

POSN PLOT UNPLOT LINE UNLINE
FILL EMPTY

Character Graphics Words

PUTBLK CHRSIZE BLKSIZE UNBLK

Three-Dimensional Graphics Words

SCREEN DRAW SEQUENCE UNDRAW AUTODRAW
OBJECT OBJADR OBJERASE OBJCOLOR SCALE
SCALX SCALY SCALZ XPOS YPOS
XTRAN YTRAN ZTRAN XROT YROT
ZROT OFF

Miscellaneous Words

CALL PREG AREG XREG YREG
. ' NOTE VOICE BELL

APPENDIX B: TECHNICAL DATA

Table of Contents

GraFORTH Memory Map

0 to 255	$0000 to $00FF	6502 Page Zero. See Page Zero listing below.
256 to 511	$0100 to $01FF	6502 Stack
512 to 767	$0200 to $02FF	GraFORTH Line Input Buffer
768 to 811	$0300 to $032B	3-D Matrix scratch-pad area
812 to 935	$032C to $03A7	Compiler Stack, PAD String Area
936 to 975	$03A8 to $03CF	Graphics Horizontal Color Buffer
976 to 1023	$03D0 to $03FF	DOS Link Area
1024 to 2047	$0400 to $07FF	Text Display Screen (used for graphics also)
2048 to 2815	$0800 to $09FF	Primary character set storage area
2816 to 5887	$0800 to $16FF	>>> User Free Space <<<
5888 to 6655	$1700 to $19FF	Image position and rotation data (See the Image Data listing below.)
6656 to 7679	$1A00 to $1DFF	Graphics address lookup tables
7680 to 7935	$1E00 to $1EFF	Data stack
7936 to 8191	$1F00 to $1FFF	Return stack
8192 to 16383	$2000 to $3FFF	Graphics screen 0
16384 to 24575	$4000 to $5FFF	Graphics screen 1
24576 to -32256	$6000 to $8200	GraFORTH System as delivered (Address approximate)

GraFORTH Memory Map

Without Language Card

-30720 to -28673	$8800 to $8FFF	Text editor file area (when used)
-28972 to -26113	$9000 to $99FF	Text editor program (when used)
-26114 to -16385	$9A00 to $BFFF	DOS 3.3
-16384 to -12289	$C000 to $CFFF	Apple][hardware I/O
-12288 to -1	$D000 to $FFFF	Apple][ROM area (Basic, Monitor)

With Language Card

-30720 to -18945	$8800 to $B5FF	Text editor file area (when used)
-18944 to -16385	$8600 to $8FFF	Text editor program (when used)
-16384 to -12289	$C000 to $CFFF	Apple][hardware I/O
-12288 to -1	$D000 to $FFFF	DOS 3.3 and Monitor

GraFORTH Page Zero Map

```
000-031 ($00-1F) not used
032-079 ($20-4F) Apple ][ monitor use
080     ($50)    GraFORTH text     pointer 1 (2 bytes)
082     ($52)    GraFORTH text     pointer 2 (2 bytes)
084     ($54)    GraFORTH graphics pointer 1 (2 bytes)
086     ($56)    GraFORTH graphics pointer 2 (2 bytes)
096-127 ($60-7F) not used (some DOS uses)
128-255 ($80-FF) used by GraFORTH
```

Useful locations in Page Zero:

128	($80)	last plotted X position
130	($82)	last plotted Y position
156	($9C)	pointer to data stack
157	($9D)	pointer to return stack
218-255	($DA-FF)	page zero matrix work area

Image Data Map

There are three data sets:

5888	$1700	undraw
6144	$1800	interim
6400	$1900	draw

Each data set contains 16 data tables, one for each of the 16 possible objects. Each data table is 16 bytes long:

Function	Relative Byte
Flag (draw, nodraw)	0
XROT	1
YROT	2
ZROT	3
XTRAN	4
YTRAN	5
ZTRAN	6
XPOS	7
YPOS	8
SCALX	9
SCALY	10
SCALZ	11
OBJCOLOR	12
Image Address	13 and 14

Each table begins at a multiple of 16. Therefore to find the object color for object 3:

16 * 3 (object 3) + 12 (object color offset)
+ 6400 (data table base address) = 6460

Three-Dimensional Mathematical Method

The three-dimensional display method used in GraFORTH][uses a system of matrices that are successively multiplied to provide the ultimate position for each line in the displayed image.

In the following diagrams, (X) through (Z) refer to rotation angles, and X through Z refer to cartesian scalar values.

Matrix 1:

Scale X	0	0
0	Scale Y	0
0	0	Scale Z

Matrix 2:

1	0	0
0	COS(X)	-SIN(X)
0	SIN(X)	COS(X)

Matrix 3:

COS(Y)	0	-SIN(Y)
0	1	0
SIN(Y)	0	COS(Y)

Matrix 4:

COS(Z)	-SIN(Z)	0
SIN(Z)	COS(Z)	0
0	0	1

This matrix transformation occurs once per image. Then the result matrix is used to transform each line position using this last matrix:

X←XTRAN	Y←YTRAN	Z←ZTRAN
0	0	0
0	0	0

After this, if a nonzero value has been selected for SCALZ, a perspective computation is made (in which case image drawing is about 20% slower). The plotting coordinates then are offset by the user-provided XPOS and YPOS values, and the line is drawn.

Image Table Format

There are four bytes for each line entry in the 3D data table. Three of these bytes are one-byte signed numbers having a range of -128 to 127, and one byte contains data about color and whether to position or draw a line:

For each entry,

Byte 1 bit 7 (high bit) is set if a line is to be drawn, clear otherwise. Bits 0-2 contain a color number 0-7 (if zero, no color change). Use of zero is recommended, this makes it possible to control image color from the program using OBJCOLOR.

Bytes 2-4 are X, Y, and Z positions within the 3D space.

The end of the image table is indicated by having the data byte (1) be equal to 255 ($FF).

Word Library Structure and Compilation

Each word entry in the library consists of three parts:

1. A "pointer location" containing the address of the next lower word in the word library.

2. The word name (ASCII characters with high bit set).

3. The executable machine language code for the word.

The hexadecimal numbers displayed by $LIST are the addresses of the pointer locations. A number returned by tic (') is the address of the executable portion of a word.

During compilation, GraFORTH separates the input line by spaces into individual words, then searches through the library for each word. For each word search, GraFORTH first reads the current value of PRGTOP to find the top of the word library. It then looks here to find a pointer containing the address of the top word within the word library. Beginning with this first word, it follows the pointers from word to word down through the library. At each word, a check is made to see if this is the word being searched for. If the word is not found, the search falls through to a routine which attempts to convert the word into a number. If this routine fails, the "Not Found" error is given.

Program lines are compiled directly into 6502 machine language in the memory immediately above the top of the word library. If the line is an "immediate" command, and not part of a word definition, the machine language code is executed, then promptly forgotten. If the line is part of a word definition, the code is saved, not executed, and the word library expands.

At execution time, calls to other words are made through direct machine language jumps. This is a major factor in the speed of GraFORTH.

Appendix C: Disk File Directory

TYPE	FILENAME	LENGTH	REMOVE OK?
B	OBJ.FORTH	36	NO
B	OBJ.EDITOR1	11	YES, IF 64K
B	OBJ.EDITOR2	11	YES, IF 48K
T	CHAREDITOR	21	NO
T	IMAGEDITOR	24	NO
T	PROFILE	15	NO
T	TURTLE	4	NO
T	PLAY	22	NO
T	STRING WORDS	4	NO
B	CHR.SYS	5	NO
B	CHR.STOP	5	YES
B	CHR.SLANT	5	YES
B	CHR.BYTE	5	YES
B	CHR.GOTHIC	5	YES
B	CHR.STUFF	5	YES
B	CHR.MAXWELL	5	YES
T	QUERY	2	YES - DEMO
T	HEADER	17	YES - DEMO
T	MENU	8	YES - DEMO
T	GRAPHICS1	8	YES - DEMO
T	GRAPHICS2	8	YES - DEMO
T	GRAPHICS3	10	YES - DEMO
T	TEXTDEMO	12	YES - DEMO
T	FORTHDESC	17	YES - DEMO
T	FLEDERMAUS	12	YES - DEMO
T	PIANO	11	YES - DEMO
T	CLOCK	5	YES
B	TETRA	2	YES - 3D
B	XYZ	2	YES - 3D
B	BAT	2	YES - 3D
B	CUBE	2	YES - 3D
B	HOUSE	2	YES - 3D
B	CHAL	10	YES - PROFILE
T	BIGCHAL	3	

Appendix D:
ASCII Characters & Equivalent Numbers

Set High Bit		Clear		
DEC	HEX	DEC	HEX	CHAR
128	80	0	00	CONTROL-@
129	81	1	01	CONTROL-A
130	82	2	02	CONTROL-B
131	83	3	03	CONTROL-C
132	84	4	04	CONTROL-D
133	85	5	05	CONTROL-E
134	86	6	06	CONTROL-F
135	87	7	07	CONTROL-G (Bell)
136	88	8	08	CONTROL-H (Left Arrow)
137	89	9	09	CONTROL-I
138	8A	10	0A	CONTROL-J
139	8B	11	0B	CONTROL-K
140	8C	12	0C	CONTROL-L
141	8D	13	0D	CONTROL-M (Return)
142	8E	14	0E	CONTROL-N
143	8F	15	0F	CONTROL-O
144	90	16	10	CONTROL-P
145	91	17	11	CONTROL-Q
146	92	18	12	CONTROL-R
147	93	19	13	CONTROL-S
148	94	20	14	CONTROL-T
149	95	21	15	CONTROL-U (Right Arrow)
150	96	22	16	CONTROL-V
151	97	23	17	CONTROL-W
152	98	24	18	CONTROL-X
153	99	25	19	CONTROL-Y
154	9A	26	1A	CONTROL-Z
155	9B	27	1B	ESCape
156	9C	28	1C	Reverse Slash
157	9D	29	1D	]
158	9E	30	1E	Up Arrow
159	9F	31	1F	
160	A0	32	20	SPACE
161	A1	33	21	!
162	A2	34	22	"
163	A3	35	23	#
164	A4	36	24	$
165	A5	37	25	%
166	A6	38	26	&
167	A7	39	27	'
168	A8	40	28	(
169	A9	41	29	)

Set High Bit DEC	Set High Bit HEX	Clear DEC	Clear HEX	CHAR
170	AA	42	2A	*
171	AB	43	2B	+
172	AC	44	2C	,
173	AD	45	2D	-
174	AE	46	2E	.
175	AF	47	2F	/
176	B0	48	30	0
177	B1	49	31	1
178	B2	50	32	2
179	B3	51	33	3
180	B4	52	34	4
181	B5	53	35	5
182	B6	54	36	6
183	B7	55	37	7
184	B8	56	38	8
185	B9	57	39	9
186	BA	58	3A	:
187	BB	59	3B	;
188	BC	60	3C	<
189	BD	61	3D	=
190	BE	62	3E	>
191	BF	63	3F	?
192	C0	64	40	@
193	C1	65	41	A
194	C2	66	42	B
195	C3	67	43	C
196	C4	68	44	D
197	C5	69	45	E
198	C6	70	46	F
199	C7	71	47	G
200	C8	72	48	H
201	C9	73	49	I
202	CA	74	4A	J
203	CB	75	4B	K
204	CC	76	4C	L
205	CD	77	4D	M
206	CE	78	4E	N
207	CF	79	4F	O
208	D0	80	50	P
209	D1	81	51	Q
210	D2	82	52	R
211	D3	83	53	S

Set High Bit DEC	Set High Bit HEX	Clear DEC	Clear HEX	CHAR
212	D4	84	54	T
213	D5	85	55	U
214	D6	86	56	V
215	D7	87	57	W
216	D8	88	58	X
217	D9	89	59	Y
218	DA	90	5A	Z
219	DB	91	5B	[
220	DC	92	5C	Reverse Slash
221	DD	93	5D	]
222	DE	94	5E	Up Arrow
223	DF	95	5F	
224	E0	96	60	`
225	E1	97	61	a
226	E2	98	62	b
227	E3	99	63	c
228	E4	100	64	d
229	E5	101	65	e
230	E6	102	66	f
231	E7	103	67	g
232	E8	104	68	h
233	E9	105	69	i
234	EA	106	6A	j
235	EB	107	6B	k
236	EC	108	6C	l
237	ED	109	6D	m
238	EE	110	6E	n
239	EF	111	6F	o
240	F0	112	70	p
241	F1	113	71	q
242	F2	114	72	r
243	F3	115	73	s
244	F4	116	74	t
245	F5	117	75	u
246	F6	118	76	v
247	F7	119	77	w
248	F8	120	78	x
249	F9	121	79	y
250	FA	122	7A	z

Appendix E: Index

Word Name	Before	After	Page
EDIT	- - -	- - -	4-2
Loads from disk (if necessary) and runs the appropriate text editor.			
ELSE	- - -	- - -	3-27
Separates the two controlled areas in an IF - ELSE - THEN construct.			
EMPTY	x y -	- - -	6-8
Erases a rectangular area from the last plotted point to (x,y).			
ERASE	- - -	- - -	5-4
Erases both graphics screens.			
EXMODE	- - -	- - -	6-10
Causes plotted points to turn on corresponding screen locations that are off, and turn off locations that are on.			
FILL	x y -	- - -	6-8
Fills a rectangular area from the last plotted point to (x,y).			
FORGET	- - -	- - -	3-17
Truncates the GraFORTH library back to the word that follows FORGET.			
GETC	- - -	n -	5-20
Gets a single character from the keyboard, placing its ASCII value on the stack.			
GETKEY	- - -	n -	5-20
Reads the keyboard without waiting, returning an ASCII value. Values over 128 are valid. Should be followed by CLRKEY.			
GETNUM	a -	n -	5-14
Converts text string at address a into a number. Unsuccessful conversions return 0.			

Word Name	Before	After	Page
GPEEK	x y -	n -	6-12
Examines point at screen coordinates (x,y). n is nonzero if point is turned on, or 0 if point is turned off.			
GR	- - -	- - -	3-38
Reestablishes normal GraFORTH input and output, and sets the graphic display mode.			
HEX	- - -	- - -	5-22
Sets number input and output to base 16.			
HOME	- - -	- - -	5-4
Erases the screen inside the text window and sets HTAB and VTAB to the upper left corner of the window.			
HTAB	h -	- - -	5-3
Sets the column for subsequent printing.			
I	- - -	n -	3-19
Returns the current innermost loop value.			
IF	n -	- - -	3-25
If n is nonzero, words between IF and THEN (or IF and ELSE) are executed, otherwise execution continues after THEN (or between ELSE and THEN).			
INVERSE	- - -	- - -	6-9
Complements the color for all text and graphics displays (including black-on-white text).			
J	- - -	n -	3-20
Returns the loop value for the next outer loop.			
K	- - -	n -	3-21
Returns the loop value for the third outer loop.			
LINE	x y -	- - -	6-4
Draws a line from the last plotted point to (x,y).			

GRAFORTH MANUAL ERRATA

As with any manual as comprehensive as GraFORTH's, a few "bugs" managed to creep past our editors. Please make note of the following changes:

PAGE	CHANGE
3-23	The last paragraph is inaccurate. The bitwise AND of some nonzero numbers will produce a zero result. However, the word AND is usually used with number comparisons that yield a 1 or 0. If both the top stack value and the second stack value are 1 (representing "true") then the AND of the two numbers will also be 1. If either or both numbers are zero, then the AND will be zero.
3-31	The BEGIN...WHILE...REPEAT diagram has the =0 and <>0 reversed. The text for this section is correct.
4-12	The fifth paragraph should refer to Appendix B, not D.
8-17	The word OFF does not immediately erase the currently selected object. It causes the next DRAW command to erase the object, without redrawing it. Subsequent commands to the object wll redraw it. The following example erases a 3-D object: Ready 3 OBJECT OFF DRAW